北京金融科技产业联盟
BEIJING FINTECH INDUSTRY ALLIANCE

GoldenDB 分布式事务型数据库金融应用指南

(2023)

指导单位：北京金融科技产业联盟

主编单位：金篆信科有限责任公司

二零二三年十一月

目录

表目录

图目录

1. 深入了解 GoldenDB

本章介绍 GoldenDB 产品应用场景、整体架构、关键特性。

1.1. 金融应用场景

GoldenDB 分布式数据库在核心系统主机下移、构建新一代金融信息系统、建设数据库云服务平台和提升金融交易系统快速决策能力等诸多应用场景下提供关键的数据库能力支撑。

1.1.1. 实现银行核心系统主机下移

GoldenDB 分布式数据库适合于银行关键系统主机下移场景。过去银行核心系统运行在集中式架构下，使用专用硬件设备，价格高昂；受限于垂直扩容瓶颈，无法支撑海量业务处理，难以平滑扩容；硬件架构、操作系统、开发语言和编码格式陈旧，升级需要停机实施，影响业务连续性。而分布式数据库通过众多普通服务器堆叠，实现海量数据存储、高并发处理能力；支持在线扩容和灰度升级，实施过程中业务 7*24 小时不间断运行；通过多副本实现系统高可靠，满足金融核心容灾和业务连续性要求，可以完全取代银行关键系统的集中式数据库，实现核心系统构架向开放式架构转型，降低建设成本，支持银行核心未来业务发展需求。

GoldenDB 分布式数据库具有海量数据存储、高性能处理、事务强一致、数据高可靠和服务高可用等优势，已经完成国有大行、股份制银行、

城商行等银行核心系统主机下移的实施。

1.1.2. 构建新一代金融信息系统

分布式数据库适合于构建新一代金融信息系统平台。国内金融科技与互联网金融业务创新发展驱动金融业务数字化和业务创新转型，迫切需要开放、灵活、快速的应用创新底座。分布式数据库具备大规模数据存储、高并发处理能力，实现平滑扩容、灰度升级以及平台化管理能力，具备灵活开放的系统架构和开发语言，支撑金融应用快速开发和业务系统快速投产，推动金融机构业务分布式架构转型，实现信息技术驱动业务创新目标。

1.1.3. 建设数据库云服务平台

金融行业 IT 基础设施上云已成为趋势，目前有近 80%金融单位启动云化转型，在实践中在 IT 建设时就采用全栈云服务方式，组建底层 IAAS、数据库、微服务框架、运维管理工具、业务中台云平台。全栈云服务形态带来开发、部署、扩容、运维和升级等一系列好处。DBaaS 以云服务技术实现数据库标准化交付、快速供给、集中监控，提升数据库的运维能力，DBA 无需进行机房、服务器、操作系统、数据库线下部署、安装升级的复杂实施，通过一键式的云服务发放可分钟级完成实现跨机房、跨地域的部署，帮助客户优化 TCO 和降低繁重和重复的运维工作。云数据库与云管平台对接，统一纳管，具备完善的告警监控、性能监控诊断等能力，减少日

常监控、故障运维难度。可以充分利用云平台弹性伸缩的能力，包括 CPU，内存、存储等，按需建设，集约资源，需要时再快速扩容。

1.1.4. 提升金融交易系统快速决策能力

GoldenDB 分布式数据库具有良好的扩展性，适合海量数据存储和处理，实现多元全量数据高效存储、不需要进行数据同步和采集，在同一个数据库内完联机处理与联机分析混合处理，在线生成数据分析结果，适合对联机交易和联机分析都有较强需求的金融业务场景，可用于实时风控、反欺诈和实时经营分析等实时数据分析系统。

1.2. 整体架构

GoldenDB 是自主研发的分布式数据库系统，整体由计算节点、数据节点、全局事务管理器、管理节点四种核心模块组成。外围包含导入导出、备份恢复等运维工具。系统采用高可靠性设计，无单点故障。计算节点为无状态多节点部署，数据集群内由多个安全分片组构成，每个安全分片组内数据节点主备多机部署，全局事务管理器主备多机部署。

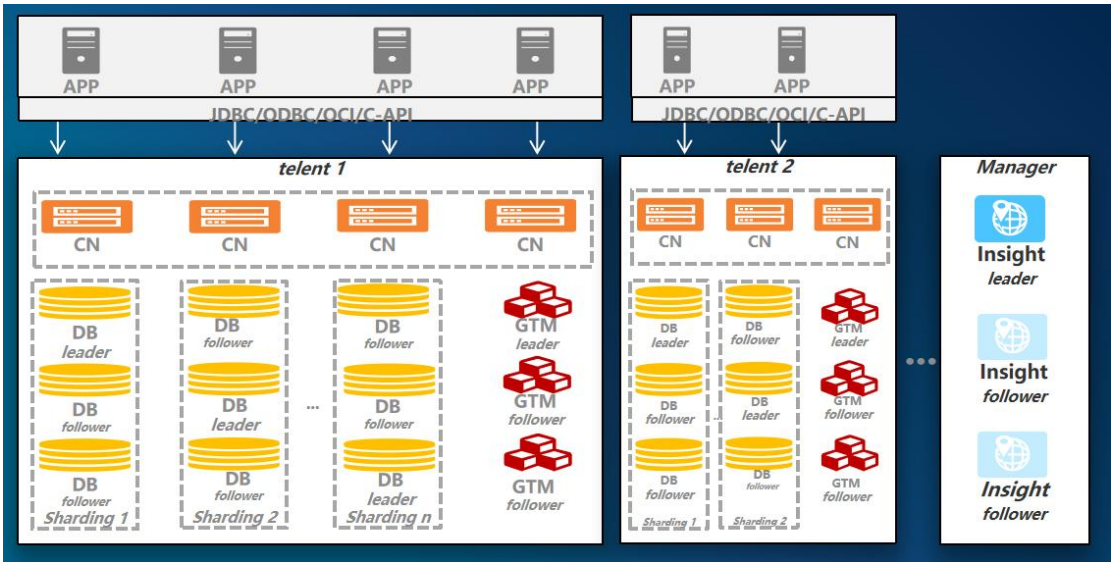


图 1 GoldenDB 逻辑架构图

计算节点（CN）。计算节点集群是分布式数据库的核心层，由无状态的计算节点组成。计算节点从应用层驱动或者管理节点接收 SQL 请求（结构化查询语言 Structured Query Language），进行逻辑优化和物理优化，生成满足分布式事务一致性的分布式查询计划；在执行分布式查询计划时，通过持续地访问数据节点，完成 SQL 请求的最终计算。

数据节点(DB)。数据节点集群是数据的最终存储模块，由多个安全组组成，每个表中的数据按照特定策略进行横向分片后存放到对应的安全组中。分片策略包括复制策略、哈希策略、范围策略、列表策略、多级分片。安全组是由一个或多个数据节点构成的数据库节点组，组内的数据库节点拥有相同的数据。当安全组中存在多个数据节点时，其中一个数据节点为主用节点，其他数据节点都为备用节点，数据在主备节点之间实时复制。主用节点具备读写能力，备用节点可以提供读能力。安全组内的数据节点

数量越多，可靠性就越高。

全局事务管理器(GTM)。全局事务管理器在分布式数据库中维护全局事务的全生命周期，提供申请、释放、查询全局事务的能力，采用双活方式部署。

管理节点(Insight)。管理节点在分布式数据库中负责集群管理流程，不涉及业务的访问流程，是GoldenDB分布式数据库产品的统一操作维护入口。用户可以在Insight上进行用户和权限管理、元数据管理、计算节点管理、数据节点管理、DDL执行、节点扩容、备份恢复、系统安装、统计及告警管理等。

1.3. 关键特性

中兴通讯在文件数据库、内存数据库和分布式关系型数据库有多年研发积累，于2014年正式立项GoldenDB分布式数据库并发布第一个版本，面向金融应用场景，历经近10年版本迭代，目前已经发展到GoldenDB7.0，推出云数据库、混合负载和兼容生态等重要特性。

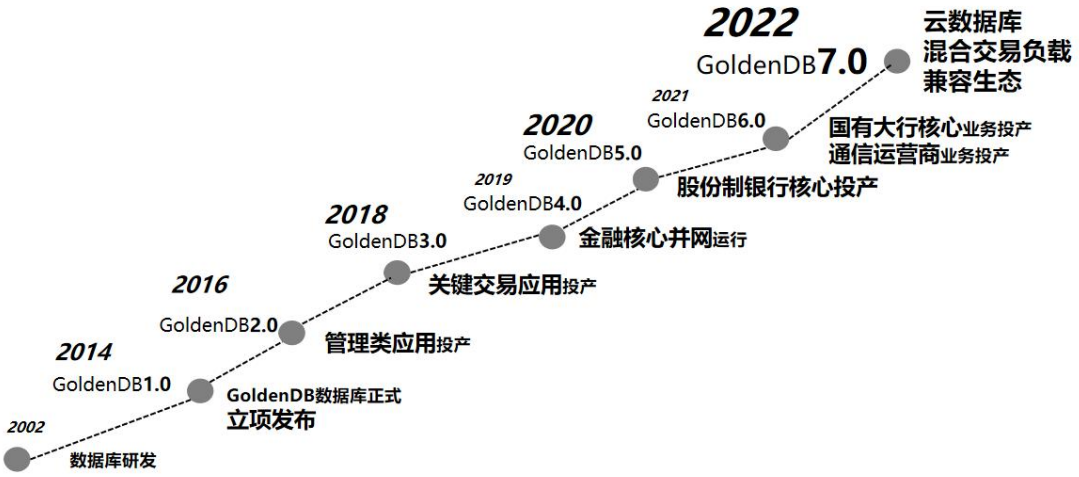


图 2 GoldenDB 发展历程

1.3.1. 在线交易（OLTP）能力

GoldenDB 分布式数据库具有如下特性：

扩展能力。GoldenDB 支持多种分片规则，包括哈希、范围、列表、复制、多级分片等。可以根据业务数据特征，选择最适合的分片规则，把数据存储在多个数据安全组中，发挥分布式数据库的最佳性能。GoldenDB 软件架构分层设计，计算节点、数据节点均可横向线性扩展，满足性能及容量的无限扩展需求。GodlenDB 支持动态扩容，扩容不需要业务停机。

分布式事务强一致性。GoldenDB 分布式数据库解决了分布式事务一致性问题，应用无需处理节点内部状态，专注于业务本身，提升了应用的开发效率和开发质量。分布式事务处理机制对应用透明，基于传统集中式数据库开发的已有应用，迁移到 GoldenDB 分布式数据库无需做事务模型改造，已有多年积累的应用资产得到继承。

高可用。GoldenDB 支持多地多中心组网，不存在单点故障，可以支持多种组网架构，独创 gSync 技术提高了数据同步性能，支持策略灵活配置，满足不同业务的可用性和可靠性要求。支持机房级故障自动切换、跨机房复制带宽限流、日切卸数、孤岛演练。

备份恢复。备份恢复是最常用的数据库日常运维手段。GoldenDB 备份恢复功能简单易用，并且保证数据恢复的一致性。GoldenDB 支持 COS、S3、NFS 等备份介质，支持集群级备份恢复，支持指定路径备份，支持压缩备份，支持数据压缩，支持第三方备份恢复工具(NBU、COS 平台)。

读写分离。充分利用系统中的备机资源，提供丰富的读写分离策略，可以指定不同机房的读写权重。

1.3.2. 混合负载（HTAP）能力

GoldenDB 自主研发分布式 SQL 引擎，重点突破分布式并行执行框架、复杂查询改写、行列混合存储、向量化等关键技术，实现一套引擎同时支撑在线交易和在线分析,避免在传统架构中在线与离线数据库之间大量的数据交互。通过行列存储、冷热数据分级存储、向量计算、LLVM, CPU 指令集优化、Online Schema Change、B+Tree 索引无锁优化等技术，大幅提升面向复杂查询场景的处理能力。

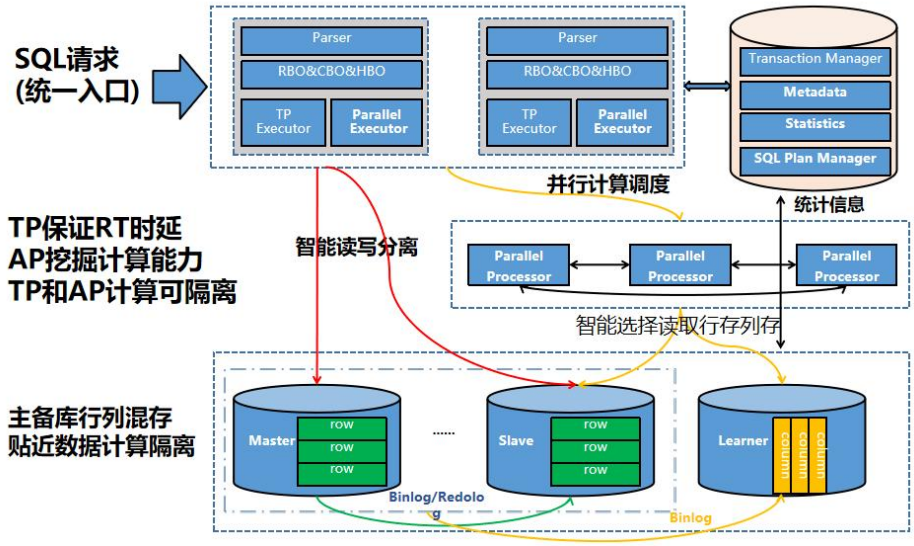


图 3 GoldenDB HTAP 关键技术

1.3.3. 云化支持（DBaaS）能力

GoldenDB 支持多种云服务形态，在 GoldenDB 数据库内部控制租户资源使用，也支持虚拟化技术部署。值得一提的是，GoldenDB 支持基于容器化交付云数据库服务，与客户的 PasS 平台融合，做到对 PaaS 平台不侵入。同时通过本地存储、物理网络，以及 GoldenDB 的数据库多副本技术，节点自愈、故障隔离能力，保障数据库云服务的高性能和高可靠。

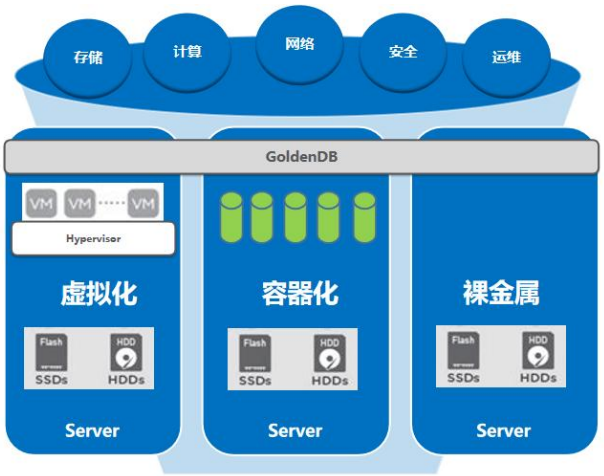


图 4 GoldenDB Cloud

GoldenDB 7.0 借助云技术来实现 DBService，从而实现数据库的集中管控、快速部署、高效运维，以及弹性扩展。GoldenDB 的运维相对集中独立，客户可以基于这套系统实时了解数据库运行的状态，能够随时洞察到当前 SQL 变化情况，根据需要做平滑的系统扩缩容。GoldenDB 云数据库具备如下特点：1）融合 采用专用存储服务器，集成软件定义存储、虚拟化、轻量级 SDN、安全等软件，实现计算、存储、网络、安全的融合；2）弹性 通过增加节点，即可快速实现容量性能弹性扩展；3）可靠 计算集群高可用，存储多副本、纠删码技术，保障系统安全可靠，支持跨机房、多地多中心部署；4）高效：环境标准化快速部署，支持全 SSD 以及 SSD 缓存+HDD 模式，提供基于裸金属、容器、虚拟化技术的实现方案；5）能效：计算成本节约 40%，存储成本节约 30%，性能提升 20%。

1.3.4. 语法兼容

GoldenDB 兼容 SQL92、SQL99、SQL2003 标准语法，完全兼容 MySQL 语法，兼容常用 Oracle、DB2 语法。

MySQL 兼容：兼容 MySQL5.7，MySQL8.0 语法和功能、兼容 MySQL 原生驱动。

Oracle 兼容：兼容 Oracle 系统视图、函数、序列、包等常用语法和高级功能。支持 ROWNUM、支持触发器、支持物化视图。

支持多级分片、支持数据透明路由(哈希、范围、列表、复制表)，支持 UPDATE 分片键，支持分布式全局唯一索引、支持分布式存储过程、支持自定义函数和视图、支持临时表。

分布式 SQL 优化。内置大量的优化规则，具备良好的复杂 SQL 语句兼容性和处理性能；支持 PREPARE 预编译、执行计划缓存、数据集透传等功能，在保证数据一致性条件下实现高性能 SQL 处理。

支持 ODBC/JDBC 接口、C/OCI 接口。

支持分布式 MVCC。

支持应用层 XA 接口。

支持 COPY TABLE。

支持闪回。

支持分布式事务死锁自动检测和释放。

支持访问黑名单/白名单控制、支持静态库功能、支持口令的国密加

密、支持高危 SQL 黑名单控制、SQL 连接请求防暴力攻击。

1.3.5. 运维管理

发布运维管理平台 Insight，实现图形化的一键式运维管理，包含租户管理、资源管理、统计监控、告警管理、权限管理、任务管理等多种运维模块，提供容灾、备份、恢复、监控、迁移、一键升级等全套解决方案。

监控统计。支持服务器资源监控(组件状态监控管理、慢 SQL 分析、日志分析、集群配置一致性比对)。提供丰富的监控指标项，对数据库租户维度、节点维度的性能、会话、SQL 等指标，进行统计和诊断，并通过图表可视化的展现,帮助用户最大限度地发现数据库存在或潜在的健康问题，全面了解 GoldenDB 数据库的使用状况。

集中告警功能，及时发现实例的异常情况，触发事件预警、告警，确保 GoldenDB 数据库稳定高效的运行。支持第三方告警平台对接接口。

支持 GoldenDB AWR 报告、命令行管理工具(DBTOOL)、支持慢 SQL 分析、支持限流控制、支持数据一致性比对、支持灰度升级、发布独立的巡检平台。

1.3.6. 平滑迁移

GoldenDB 提供一整套迁移辅助工具，实现存量应用从传统集中式数据库快速平滑迁移到分布式数据库。

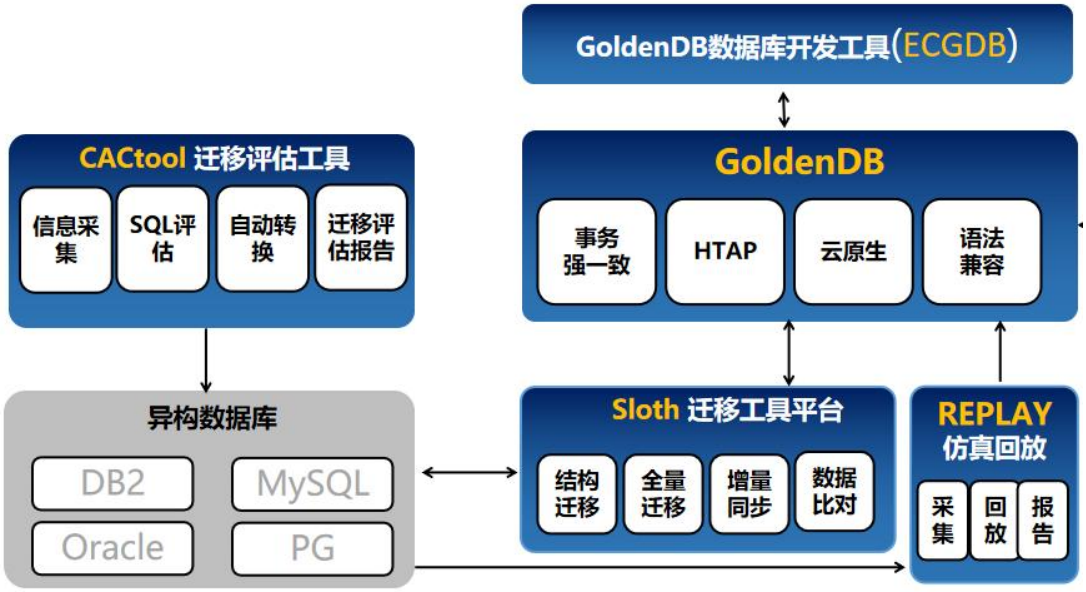


图 5 GoldenDB 数据迁移工具集

迁移评估工具(CACtool)：CACtool 工具通过简单配置即可访问异构数据库，采集库表结构、各种数据库对象以及过往 DML 语句，导出适配 GoldenDB 语法对象，在导出过程中自动进行转换，给出迁移可行性和兼容性客观详实分析，包含配置方案(IO、网络、CPU)，应用改造要求等，并形成建议评估报告。

迁移平台化工具 Sloth：支持存量应用的全量迁移、增量同步和数据比对。迁移数据的准确性很重要，将数据从源库以全量或增量方式迁移过来后，要验证迁移数据的正确性，Sloth 可以快速校验、多批次执行，通过可视化页面进行迁移任务管理，大大减少了人工执行工作量，避免应用开发校验工具，减少出错。Sloth 工具还具备反向同步能力，即应用从存量数据库迁移到 GoldenDB 后，将 GoldenDB 上产生的增量数据实时同步回

原数据库。

Replay 工具：在 GoldenDB 正式接替存量数据库投产前，通过实时采集存量数据库上的 WCR 文件，将其还原为 SQL 语句，向 GoldenDB 进行回放，通过这种方式，使用来自生产上真实的 SQL 来验证 GoldenDB 数据库执行的正确性，同时也通过生产压力对 GoldenDB 处理性能和稳定性进行充分验证。

1.3.7. 全栈国产化

信创方面，支持国产化操作系统，包括统信 UOS、麒麟软件、中兴新支点、方德、深之度、凝思等。

支持国产化芯片,包括鲲鹏、海光、龙芯、兆芯、飞腾等。

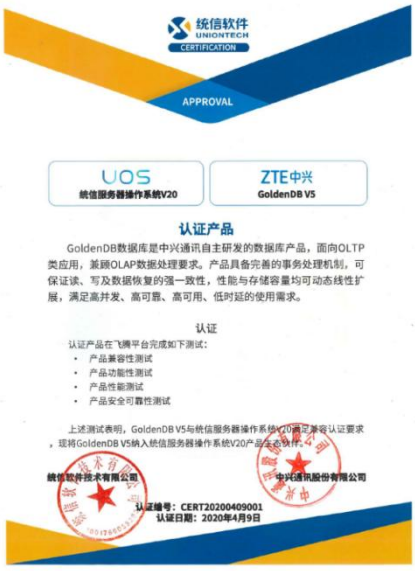


图 6 GoldenDB 兼容操作系统、芯片认证证书

1.4. 路标规划

未来 3 年，GoldenDB 继续紧贴金融行业的数据库应用需求，在内核架构、语法兼容性、处理性能及智能运维四大部分发力，坚持自主研发，持续完善产品能力：

关键特性	2023	2024	2025
内核架构	HTAP：降低数据存储成本、强化分析处理引擎； 内核增强：智能资源管理、多模SQL引擎；支持云原生多租户等。	增强分布式优化器优化性能：增强基于统计和代价计算的优化算子；重构 OCI 架构。	推出高性能安全数据库：支持全密态数据库、表防篡改、表追溯能力、动态脱敏和透明加密，软硬结合的加解密架构。硬件加速：基于 FPGA 板卡加速数据

			库计算。
语法兼容性	全面兼容 ORACLE、DB2 语法，包括系统视图、函数、包、PL/SQL 等高级功能。	全面兼容异构数据库，实现 Oracle/DB2/ MySQL 端到端自动化迁移流水线工具链能力构建。	兼容 PG，加强自动化迁移流水线工具链易用性和平台化管理能力。
处理性能	提升读写性能，支持并行能力，支持执行计划绑定；优化 PREPARE 机制，插入性能提升 20%。	进一步优化后端 PREPARE 能力；基于 redo 日志提升同步与回放性能； 采用新的分布式事务机制，优化单分片事务处理性能。	优化 CN 与 DN 间内部接口，融合 CN 和 DN 的处理过程，减少内部网络传输和处理； 云数据库：优化上云性能，完成上云产品，基于国资公有云发布。
智能运维	完善数据库迁移和运维工具，实现迁移评估工具、仿真回放工具，增强并推出独立的故障定位和诊断工具。	自动化高效运维：自动化扩缩容、历史 SQL 比对、自动化滚动升级，故障智能分析。	AI 智能运维：基于 AI 技术实现自感知、自诊断、自调优、自驱动。

2. GoldenDB 金融应用规划

描述在金融应用规划阶段，做好关键资源要素设计，包括：服务器规划、高可用设计和容灾方案。

2.1. 物理机规划

2.1.1. 选择合适的服务器

硬件配置，推荐 CPU、内存、磁盘典型规格，RAID 配置、网络冗余。

表 1 典型服务器配置规格

服务器型号	支持：海光服务器，鲲鹏服务器，Intel 服务器等。
CPU	标准推荐：2 路 24C 以上，支持：Intel X86、海光 X86、鲲鹏、龙芯、飞腾、兆芯等。
MEM	标准推荐：128G 以上，典型配置 256G、384G 或 512G。
DISK	系统盘：2*480，做 RAID1 数据盘：2 块 1.92T 以上，推荐 SAS SSD 磁盘以上，做 RAID10。
网络	万兆网络，双网卡绑定。
OS	支持 KylinV10sp2/sp3，Redhat7.6 /Centos7.6 以上，统信 UOS、中兴新支点、方德、深之度、凝思、SUSE、Redhat。

GoldenDB 为纯软件组件构成，支持虚拟部署，最少要求 3 台 12C 16G 500G 硬盘(管理节点、数据节点、计算节点合设)。

2.2. 数据库节点设计评估因素

在做数据库节点设计时，需要关注一下几个关键因素：

数据量 单机数据量超过 1TB，有必要开始考虑使用分布式数据库；超过 2TB 的数据量，需要使用多分片设计。数据量的评估不仅需要考虑当前存量，还要考虑未来的增量。

交易量 业务系统 TPS/QPS 超过 2000TPS/20000QPS，有必要开始考虑使用分布数据库；超过 4000TPS/40000QPS，需要考虑使用多分片。交易量评估需要考虑：当前交易量、预期交易增量。结合平均 TPS 和峰值 TPS 两种情况。

2.3. 数据库节点评估方法

数据库节点数量依据总数据量、单服务器存储容量、总 TPS 要求、单分片 TPS 预估共同确定。

2.3.1. 以数据量作为节点数计算方式

第一步计算单台服务器的存储容量，服务器磁盘需要做 RAID，推荐采用 RAID10,这样服务器的总存储容量是裸容量的 1/2，亦可以采用其他 RAID 模式。一般服务器磁盘需要预留一定的空闲空间(10~30%)。如以下服务器为例，使用 RAID10 后，预留 30%空闲空间，则单台服务器容量为 9.4T。

表 2 服务器存储容量计算示例

单块容量(T)	磁盘数量	裸容量(T)	RAID10(T)	空闲空间	**单台服务器容量(T)**
1.92	14	26.88	13.44	30%	**9.4**

第二步计算业务存储容量需求。假设设计规模为 38T，数据库日志空

间为 4T，计算得到总数据库空间为 42T。

表 3 数据总量计算示例

业务	设计规模(T)	日志规模(T)	**总数据库空间(T)**
XX 系统	38	4	**42**

第三步根据副本数量计算总的存储节点规模。

表 4 通过数据量计算服务器节点数示例

业务	总空间(T)	单台容量(T)	单副本台数	副本数量	存储节点数(台)
XX 系统	42	9.4	42/9.4~=5	3	15

2.3.2. 以处理性能作为计算方式

数据节点：以业务的 TPS 和 QPS，按每台机器 4000 TPS 或者 40000 QPS 相除(2 路 24C+384G+SSD)，得到数据分片数量。

计算节点：业务的 TPS 和 QPS，按每台机器 6000 TPS 或者 60000 QPS 相除(2 路 24C+384G)，得到计算节点数量。

2.3.3. 机器数量计算结果

数据节点机器数量=max（总数据量/单分片数据量，总 TPS 要求/单分片 TPS 预估）*副本数量。

计算节点机器数量=总 TPS 要求/单计算节点 TPS 预估。

2.3.4. 资源评估工具示例

根据上述资源评估计算方法，可整理输出资源评估的小工具/公式（excel 工具），具体如下：

数据节点(按存储规模计算)

表 5 按存储规模计算数据节点数量

参数	计算公式	说明
单块容量(T)	1.92	输入值，根据服务器配置给定，影响单台服务器存储容量。
磁盘数量	14	输入值，根据服务器配置给定，影响单台服务器存储容量。
RAID 空间折算	1/2	输入值，RAID0 为 1，RAID10 为 1/2，RAID5(每 4 块磁盘)为 3/4。生产系统推荐 RAID10。
空闲空间比率(%)	30%	输入值，预留一定的磁盘空闲空间。推荐 30%。
单台服务器容量(T)	9.4	计算值，单块容量*磁盘数量*RAID 空间折算*(1-空闲比例)计算得出。
设计规模(T)	38	输入值，包括表数据和索引。
数据库日志空间(T)	4	输入值，给数据库日志的存储空间，以每日生产数据库日志和天数进行计算，并建议至少保留 7 天。
总数据库空间(T)	42	计算值，设计规模+数据库日志空间。
数据分片数量（存储规模）	5	计算值，总数据库空间/单台服务器容量。

数据节点(按计算规模计算)

表 6 按计算规模计算数据节点数量

参数	计算公式	说明
应用 TPS 峰值	24000	输入值，实测或统计得出。
应用 QPS 峰值	240000	输入值，实测或统计得出。
单节点 TPS 性能	4000	经验值，与 CPU/内存/存储有关，基准为 2 路 24C+384G 内存+SSD 存储。
单节点 QPS 性能	40000	经验值，与 CPU/内存/存储有关，基准为 2 路 24C+384G 内存+SSD 存储。
TPS 所需台数	6	计算值，应用 TPS 峰值/单节点 TPS 性能。
QPS 所需台数	6	计算值，应用 QPS 峰值/单节点 QPS 性能。
数据库节点台数(计算规模)	6	计算值，TPS 和 QPS 所需台数取最大值。

计算节点(按计算规模计算)

表 6 按计算规模计算数据节点数量

参数	计算公式	说明
应用 TPS 峰值	24000	输入值，实测或统计计算得出。
应用 QPS 峰值	240000	输入值，实测或统计计算得出。
单节点 TPS 性能	6000	经验值，与 CPU/内存有关，基准为 2 路 24C+384G 内存。
单节点 QPS 性能	60000	经验值，与 CPU/内存有关，基准为 2 路 24C+384G 内存。
TPS 所需台数	4	计算值，应用 TPS 峰值/单节点 TPS 性能。

QPS 所需台数	4	计算值，应用 QPS 峰值/单节点 QPS 性能。
计算节点台数（计算规模）	4	计算值，TPS 和 QPS 所需台数取最大值。

数据分片按数据量计算为 5 台，按处理性能计算为 6 台，两者取最大值，为 6 台。

服务器台数合计

表 7 服务器总数量计算过程

数据节点台数	**6*3**	计算值，取数据节点台数(存储规模)和数据节点台数(计算规模)最大值，再乘以三副本。
计算节点台数	4	计算值，TPS 和 QPS 所需台数取最大值。
管理节点台数	**1*3**	经验值，GoldenDB Insight 和 GTM 合设，三个副本。
GoldenDB 总服务器台数	25	三种节点合计。

2.3.5. 容器化方式部署

GoldenDB 也支持容器化部署方式。采用容器化部署时，根据应用的 TPS/QPS 选择合适的容器资源。具体如下：

1） CN 节点。DN 节点提供两种配置 8C32G、16C64G。不同配置的性能容量如下：

表 8 计算节点典型容器化配置规格

	CPU	内存	TPS	QPS
CN	8C	32G	1500	15000
	16C	64G	3000	30000

CN 个数=总 QPS 需求/CN 的 QPS。

出于高可用设计要求，CN 个数不得少于 2 个。

2） DN 节点。 DN 节点提供三种配置 8C32G、16C64G、16C128G。不同配置的性能容量如下：

表 9 数据节点典型容器化配置规格

	CPU	内存	单库 TPS	QPS
DN	8C	32G	1000	10000
	16C	64G	2000	20000
	16C	128G	2000	20000

推荐单个 DN 的数据量不超过 500G，根据 QPS 需求、数据量计算分片个数，再根据应用高可用需求计算 DN 容器数，推荐三副本以上。

分片个数=MAX((向上取整)总 QPS 需求/DN 的 QPS,(向上取整)数据总量/500)。

DN 个数=分片个数乘以副本数。

1) GTM 节点。GTM 节点的配置为 4C16G，租户级 GTM，高可用与 DN 对等，推荐三副本以上。

2.4. 业务连续性规划

应基于《商业银行业务连续性监管指引》以及金融单位自身的业务连续性管理办法，确定具体应用的业务连续性目标、设计对应的高可用方案。

GoldenDB 支持单数据中心高可用、同城双中心、两地三中心、多地多中心等容灾架构模式。GoldenDB 基于多副本冗余技术、一致性复制技术和灵活高可用策略实现多种容灾方案，单台服务器故障自动切换到本数据中心其他机器的副本上；双中心以上的机房故障时，可快速切换到其他机房或者城市灾难中心，最大程度保证业务连续性。

2.4.1. 单数据中心

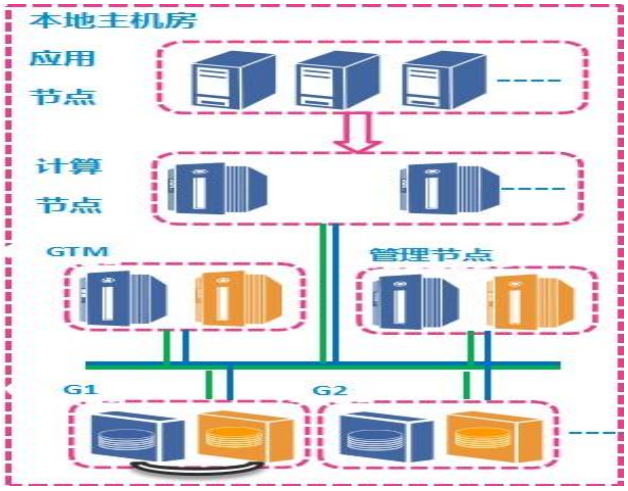


图 7 单中心高可用

单数据中心的架构适用于管理类应用，对数据安全性要求不是特别高。在本地机房内任何一个组件都不能出现单点，管理节点使用三节点来进行高可靠设计。计算节点根据应用的性能需求和可靠性需求合理配置节点数量，但节点数量不能低于 2 个。数据集群的安全组内的数据节点副本数根

据应用对数据安全性的要求进行设置，建议副本数设置为 3 个。

表 10 单数据中心典型配置

组件类型	部署建议
计算节点	根据业务规模配置，至少 2 个节点
数据节点	两个副本以上，推荐三副本。
GTM	两个副本以上，推荐三副本。
管理节点	三副本。

2.4.2. 同城双中心

同城双中心架构适用于交易类应用，除了需要进行组件级别的容灾还进行机房级别的容灾。GoldenDB 支持同城双中心的架构。推荐同城之间的网络时延需要小于 2~5ms，否则会对应用时延会产生较大影响。

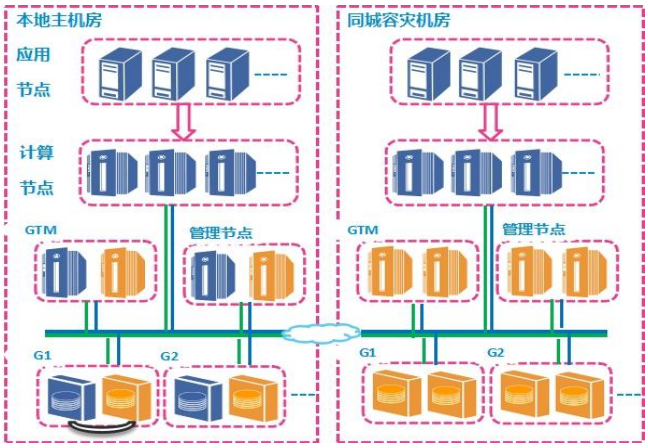


图 8 同城双中心容灾

在 GoldenDB 同城双中心架构中，管理节点 3 个或者 5 个，分布在两个数据中心中。计算节点在两个机房中的地位和作用是对等的，都可以接入应用的请求和并互为容灾备份。数据节点根据应用需求进行配置，一般

本地机房和同城机房副本数相同，比如每个机房内包括 2 个副本。

表 11 同城双中心典型配置

组件类型	部署建议
计算节点	两个机房对等部署，每个机房至少 2 个节点。
数据节点	两个机房对等部署，推荐每个机房部署 2 个副本，副本数量增加可靠性越强。
GTM	两个机房对等部署，推荐每个机房部署 2 个副本，副本数量增加可靠性越强。
管理节点	3 副本或 5 副本。

2.4.3. 两地三中心

两地三中心的架构设计适用于核心关键应用，是典型金融容灾部署方式。如果出现机房级故障，如火灾、机房停电等情况，可自动或手动切换到同城机房；如果出现大面积水灾、地震等城市灾难，通过手工方式切换到其他城市的灾备机房。

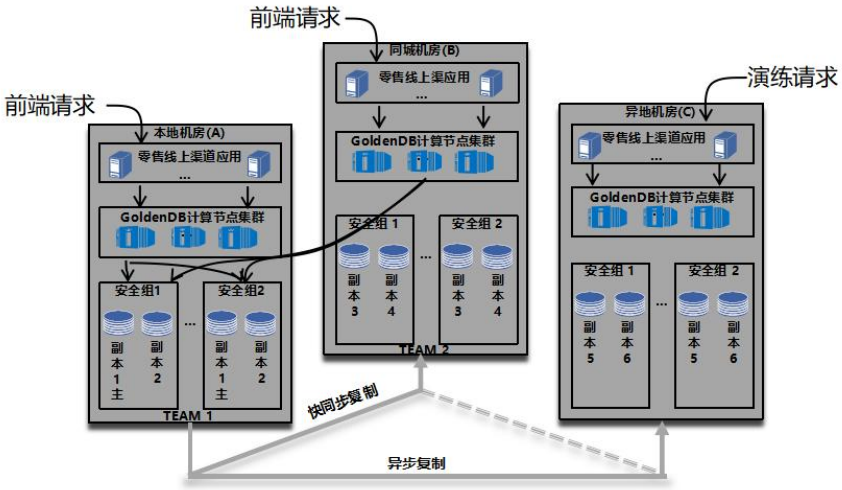


图 9 同城双活方案

GoldenDB 支持两地三中心架构。GoldenDB 在本地数据中心和同城数据中心部署等量的计算节点集群和数据副本，异地机房再部署一组 GoldenDB 副本。本地机房与异地机房之间采用异步复制方式，最大限度保证应用的可用性。

同城双活。本地同城两个数据中心的计算节点均对外提供服务，正常情况下所有数据分片主节点部署在本地机房，亦可将部分数据分片主节点动态调度到同城机房，实现同城双活。

表 12 两地三中心典型配置

组件类型	部署建议
计算节点	本地同城机房对等部署，每个机房至少 2 个节点；或根据业务量增加计算节点数量。
数据节点	本地同城对等部署，推荐每个机房部署 2 个副本，副本数量增加可靠性越强。异地一般部署 1 个副本，孤岛演练再部署 1 个副本。

GTM	本地同城对等部署，推荐每个机房部署 2 个副本，副本数量增加可靠性越强。异地一般部署 1 个副本，孤岛演练再部署 1 个副本。
管理节点	3 副本或 5 副本。

2.4.4. 多地多中心多活

多地多中心可以做到更大的可用性，如任意单个机房出现故障，无需人工干预即可进行自动切换，且切换后无数据丢失(RPO=0，RTO<30s)。总体方案如下图：

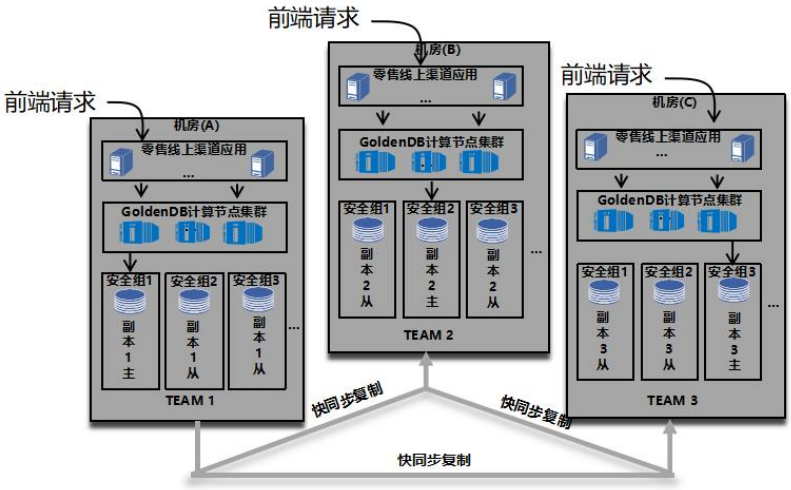


图 10 多中心多活方案

应用系统在三个机房同时部署，数据按机房打散到三个机房，前端请求根据数据所在位置路由到对应机房的应用系统，应用只处理本机房数据。GoldenDB 将三个数据中心按三个 TEAM 设置高可用策略，其中正常响应数配置为 3，可写响应数配置为 2。若其中一个数据中心出现故障，业务不

受影响，数据不会丢失。

表 13 多地多中心典型配置

组件类型	部署建议
计算节点	其中两个机房对等部署，每个机房至少 2 个节点；第三个机房部署 1 个节点;或根据业务量增加计算节点数量。
数据节点	其中两个机房对等部署，推荐每个机房部署 2 个副本，副本数量增加可靠性越强。第三个机房部署 1 个副本。
GTM	其中两个机房对等部署，推荐每个机房部署 2 个副本，副本数量增加可靠性越强。第三个机房部署 1 个副本。
管理节点	3 副本或 5 副本。

3. 基于 GoldenDB 的金融应用开发

基于 GoldenDB 的 Java 开发和集中式数据库没有本质的区别，推荐使用 GoldenDB Java 官方驱动程序，同时兼容 MySQL JDBC 驱动。

3.1. 应用开发快速入门

3.1.1. 一个 Java 应用 demo

一个简单的示例程序，如下：

```
/*
   查询示例
*/
import java.sql.*;
```

```
    public class ConnectGDB {
       public static void main(String args[]) {
          try {
                Class.forName("com.mysql.jdbc.Driver");// 加载 GoldenDB
JDBC 驱动
                System.out.println("Success loading Mysql Driver!");
             } catch (Exception e) {
                System.out.println("GoldenDB 驱动加载出错,请检查 GoldenDB
的连接！ ");
                e.printStackTrace();
             }
          try{
             Connection                                conn                =
DriverManager.getConnection("jdbc:mysql:loadbalance://10.229.xxx.xxx:6607
,10.229.xxx.xxx:7707/testdb?", "appuser", "db10$ZTE");
             System.out.println("连接 GoldenDB 数据库成功");

             PREPAREdStatement ps = null;
             ResultSet rs = null;
             ps=conn.PREPAREStatement("select * from t_area where name
= ?");

             ps.setString(1, "东城区");
             ResultSet rs=ps.executeQuery();

             System.out.println("sql 语句执行完成 1");
```

```
            while(rs.next()) {
                System.out.println(rs.getString("id"));
            }

            stat.close();
            ps.close();
            conn.close();
        } catch (SQLException e) {
            System.out.print("jdbc 连接 GoldenDB 数据库出抛出异常  或 sql
语句执行异常  详情请 debug");
            e.printStackTrace();
        }finally {
        }
    }
}
```

3.1.2. 通过 JDBC 访问 GoldenDB

JDBC 是 Java 语言中用来规范客户端程序如何来访问数据库的应用程序接口。JDBC 主要由四大功能模块组成：

1、DriverManager 类：JDBC 管理层，加载驱动程序。

2、Connection 接口：数据库连接对象。

3、Statement 接口：数据库操作对象。

4、ResultSet 接口：数据库结果集，Statement 执行 SQL 语句时返回 Result 结果集。

GoldenDB Java 驱动是对 JDBC 驱动规范的完全实现，已适配 JDBC 4.2 版本(随 JDK1.8 发布)，可按照 JDBC 规范使用。

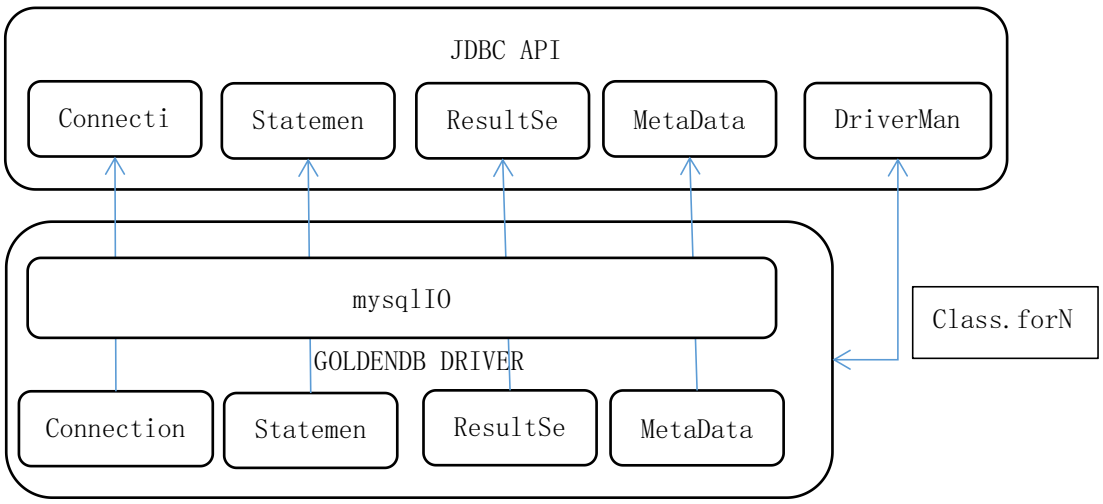


图 11 JDDB 逻辑结构图

重要功能参数:

表 14 JDBC 重要配置参数

序号	功能名称	依赖相关参数组合
1	Loadbalance 负载均衡	Loadbalance+自研协议 isConnectionLevel=true
2	PREPARE 及预编译功能	useCursorFetch=true&prepStmtCacheSqlLimit&cachePrepStmts&prepStmtCacheSize&PREPAREStrategy
3	黑名单检测功能	shadowThreadSwitch=true&intervalTime&blackTaskTime&errorCount
4	超时参数	connectTimeout&socketTimeout
5	LB 分组协议	proxygroups&minconnectionproxys&proxygroup(n)

6	新协议批量插入功能	comStmtBatchFlag=true&rewriteBatchedStatements=true 支持 insert 多 values 走新协议

3.1.3. 使用 GoldenDB LoadBalance

GoldenDB 支持“软负载”，即通过软件（驱动）实现负载均衡，可以将读写请求行负载均衡，路由到多个计算节点上，LoadBalance 协议具备 Failover 特性。它的特点是配置简单、使用灵活且成本较低。示例：

```
url=jdbc:mysql:loadbalance://10.46.178.105:1234,10.46.178.195:1234/test_sj?useSSL=false&connectTimeout=60000&socketTimeout=60000&cachePrepStmts=false&isConnectionLevel=true
```

其中，10.46.178.105:1234,10.46.178.195:1234 分别表示两个计算节点，分别对应了 1234 端口，应用侧建链时根据上述负载均衡策略将建链请求均衡下发到这两个计算节点。

3.1.4. 分组优先级

支持将计算节点按分组配置优先级,实现机房故障应用请求自动切换。可以将若干计算节点组成多个逻辑组，示例：

```
url="jdbc:goldendb:loadbalance://192.168.100.11:7788,192.168.100.12:7788/test?minconnectionproxys=1&proxygroups=2&proxygroup1=192.168.100.7:7788,192.168.100.8:7788&proxygroup2=192.168.100.9:7788,192.168.100.10:7788"
```

其中，Proxygroups 为优先级分组个数，proxygroup1，proxygroup2.....计算节点分组列表，优先级依次降低。正常只访问优先级高的计算节点，当优先级高的计算节点都异常时，则连接次优先级的计算节点。

3.1.5. 连接池配置

数据库连接池使用 javax.sql.DataSource 来表示,DataSource 知识一个接口，该接口的通常由第三方提供实现，GoldenDB 兼容适配主流的连接池产品，如 C3P0,Druid,DBCP 等。以 Druid 配置为例（备注：不同版本存在参数差异）：

```
    driverClassName=com.goldendb.jdbc.Driver
    url=jdbc:goldendb://10.218.126.208:8881/sysbench?&useCursorFetch=tru
e
    initialSize=10
    minIdle=10
    maxActive=100
    testOnBorrow=false
    testOnReturn=false
    testWhileIdle=true
    validationQuery=SELECT 1 FROM DUAL
    timeBetweenEvictionRunsMillis=30000
    minEvictableIdleTimeMillis=600000
    keepAlive=true
```

保活配置参数是 validationQuery、keepAlive。

3.1.6. 字符集选择

字符集设置支持实例级、库级、表级以列级别，字符集支持：latin1、gbk、utf8、utf8mb4 和 gb18030，对应字符序支持按照二进制值进行，即对应上述四种字符集的字符序分别为 latin1_bin、gbk_bin、utf8_bin、utf8mb4_bin 和 gb18030_bin，的不同级别设置方式参考如下：

- 实例级。通过 Insight 页面新增租户，“租户规划”阶段设置【字符集】

图 12 实例配置字符集

- 数据库级。通过建库语句指定字符集编码，如下：

```
CREATE DATABASE `db01` CHARACTER SET 'utf8mb4' COLLATE 'utf8mb4_bin';
```

- 表级。通过建表语句指定字符集编码和字符序，创建表 t01，字符集为 utf8,字符序为 utf8_bin，SQL 如下：

```
CREATE TABLE `t01` (
  `ROLE_ID` varchar(20) NOT NULL,
  `MENU_ID` varchar(50) NOT NULL,
  PRIMARY KEY (`ROLE_ID`,`MENU_ID`),
  UNIQUE INDEX `SYS_C00192546` (`ROLE_ID`,`MENU_ID`),
  INDEX `role_id_ix` (`ROLE_ID`)
) DEFAULT CHARSET=utf8 COLLATE=utf8_bin
distributed by hash(ROLE_ID)(g1,g2);
```

- 列级。调整表 t01 的 MENU_ID 列字符集为 utf38mb4,SQL 如下：

```
ALTER TABLE t01 MODIFY MENU_ID VARCHAR(255) CHARACTER SET utf8mb4;
```

3.1.7. 使用 PREPARE

SQL 语句的语法校验和执行树生成花费时间较长,GoldenDB 支持使用预编译(PREPARE)功能，可对 SQL 语句进行一次语法校验和编译，提供整体执行效率。

```
// 预处理添加数据
ps = conn.PREPAREStatement("select * from test_dev.emp where empno = ?");
ps.setInt(1, 1);
rs = ps.executeQuery();
//再次执行，对 empno 赋值不同
ps.setInt(1, 2);
rs = ps.executeQuery();
```

3.2. 通过其他方式访问 GoldenDB

3.2.1. 采用 ODBC 访问 GoldenDB

GoldenDB 支持采用 ODBC 进行访问数据库，需要安装 unixODBC 和 ZXCLOUD-GoldenDB-ALL-ODBCV1.0.1（具体版本以实际为准），兼容 mysql-connector-odbc 驱动程序。代码示例如下：

```
int main()
{
    short sret; //返回代码
    void* henv;  //环境句柄
    void* hdbc;  //连接句柄
    long mode;//提交模式
    void* hsmt; //语句句柄

    /** 申请环境句柄 **/
    sret = SQLAllocHandle(SQL_HANDLE_ENV,SQL_NULL_HANDLE,&henv);
    if(!isSuc(sret))printf("申请环境句柄出错\n");
    /** 设置环境属性,声明 ODBC 版本 **/
    sret =
SQLSetEnvAttr(henv,SQL_ATTR_ODBC_VERSION,(void*)SQL_OV_ODBC3,SQL_IS_INTE
GER);
    if(!isSuc(sret))printf("声明 ODBC 版本出错\n");
    /** 申请连接句柄 **/
    sret = SQLAllocHandle(SQL_HANDLE_DBC,henv,&hdbc);
    if(!isSuc(sret))printf("申请连接句柄出错\n");
    /** 连接数据源 **/
```

```
    sret = SQLDriverConnect(hdbc,NULL,(SQLCHAR*)szConnectString,SQL_NTS,
(SQLCHAR *)szDriverOutput,256,&sDriverOutputLength,SQL_DRIVER_NOPROMPT);
    /** 分配语句句柄 **/
    sret = SQLAllocHandle(SQL_HANDLE_STMT,hdbc,&hsmt);
    if(!isSuc(sret))printf("分配语句句柄出错\n");
    /** 执行 create 语句 **/
    printf("execute sql: %s\n\n", szSQLCreate);
    sret = SQLExecDirect(hsmt,(SQLCHAR *)szSQLCreate,256);
    if(!isSuc(sret))printf("exec create sql fail !\n\n");
    else printf("create table t1(new) success !\n\n");

    /** 执行 select 语句 **/
    printf("execute sql: %s\n\n", szSQLSelect);
    sret = SQLExecDirect(hsmt,(SQLCHAR *)szSQLSelect,256);
    if(!isSuc(sret))printf("exec select sql fail !\n\n");
    else printf("select sql exec success !\n\n");

    /** 释放语句句柄 **/
    SQLFreeHandle(SQL_HANDLE_STMT,hsmt);
    /** 提交连接的事务 **/
    SQLEndTran(SQL_HANDLE_DBC,hdbc,SQL_COMMIT);
    /** 断开与数据源的连接 **/
    SQLDisconnect(hdbc);
    /** 释放连接句柄 **/
    SQLFreeHandle(SQL_HANDLE_DBC,hdbc);
    /** 释放环境句柄 **/
    SQLFreeHandle(SQL_HANDLE_ENV,henv);
```

```
    printf("The connection will be closed !\nBye !\n");
}
```

3.2.2. 采用 OCI 访问 GoldenDB

GoldenDB 支持采用 OCI 进行访问数据库，配置 dci.ini 里面数据库 IP 地址和端口，参考示例如下：

```
    //使用线程和对象模式来创建环境句柄
    OCIEnvCreate(&myenvhp, OCI_THREADED|OCI_OBJECT, (dvoid *)0,0, 0, 0,
(size_t) 0, (dvoid **)0);
    //分配服务器句柄
    OCIHandleAlloc ((dvoid *)myenvhp, (dvoid **)&mysrvhp,OCI_HTYPE_SERVER,
0, (dvoid **) 0);
    //分配错误句柄
    OCIHandleAlloc ((dvoid *)myenvhp, (dvoid **)&myerrhp,OCI_HTYPE_ERROR,
0, (dvoid **) 0);
    //分配服务器上下文句柄
    OCIHandleAlloc ((dvoid *)myenvhp, (dvoid **)&mysvchp,OCI_HTYPE_SVCCTX,
0, (dvoid **) 0);
    //设置服务器上下文句柄的服务器句柄属性
    OCIAttrSet ((dvoid *)mysvchp, OCI_HTYPE_SVCCTX,(dvoid *)mysrvhp, (ub4)
0, OCI_ATTR_SERVER, myerrhp);
    //分配用户会话句柄
    OCIHandleAlloc ((dvoid *)myenvhp, (dvoid **)&myusrhp,OCI_HTYPE_SESSION,
0, (dvoid **) 0);
```

```
    //为用户会话句柄设置用户名和密码属性
    OCIAttrSet ((dvoid *)myusrhp, OCI_HTYPE_SESSION,(dvoid *)username,
(ub4)strlen((char*)username),OCI_ATTR_USERNAME, myerrhp);
    OCIAttrSet ((dvoid *)myusrhp, OCI_HTYPE_SESSION,(dvoid *)password,
(ub4)strlen((char*)password),OCI_ATTR_PASSWORD, myerrhp);
//在服务器上下文环境中设置用户会话属性
    OCIAttrSet ( (dvoid *)mysvchp, OCI_HTYPE_SVCCTX,(dvoid *)myusrhp, (ub4)
0, OCI_ATTR_SESSION, myerrhp);

    //drop
    //分配语句句柄
    CheckErr(myerrhp, OCIHandleAlloc(myenvhp, (void**)&stmthp_drop,
OCI_HTYPE_STMT, 0, 0));
    //准备 SQL 语句
    CheckErr(myerrhp, OCIStmtPREPARE(stmthp_drop, myerrhp, drop,
strlen((char*)drop),OCI_NTV_SYNTAX, OCI_DEFAULT));
    status = OCIStmtExecute(mysvchp, stmthp_drop, myerrhp, 1, 0, NULL,
NULL,OCI_DEFAULT);
    printf("drop table t1(old) success !\n");
    //select
    //分配语句句柄
    CheckErr(myerrhp, OCIHandleAlloc(myenvhp, (void**)&stmthp_select,
OCI_HTYPE_STMT, 0, 0));
    //准备 SQL 语句
    CheckErr(myerrhp, OCIStmtPREPARE(stmthp_select, myerrhp, select,
```

```
strlen((char*)select),OCI_NTV_SYNTAX, OCI_DEFAULT));

    status = OCIStmtExecute(mysvchp, stmthp_select, myerrhp, 1, 0, NULL,
NULL,OCI_DEFAULT);

    OCIAttrGet(stmthp_select, OCI_HTYPE_STMT, &rows_processed, 0,
OCI_ATTR_ROW_COUNT, myerrhp);

    printf("The numbers of successfully-selected rows is %d\n",
rows_processed);

    //结束会话
    OCISessionEnd(mysvchp, myerrhp, myusrhp, OCI_DEFAULT);
    //断开连接
    OCIServerDetach(mysrvhp, myerrhp, OCI_DEFAULT);
    //释放环境句柄
    OCIHandleFree((void*)myenvhp, OCI_HTYPE_ENV);
    return 0;
```

3.3. 如何进行数据均衡设计

3.3.1. 数据均衡手段

业务大表建议采用分片表、分片+分区、多级分片的方式实现数据的均衡分布，分片策略支持：哈希、范围、列表，各分片上的数据量均衡，业务增长过程中数据变化也保持相对均衡，其中

1) 面向用户的主档表(客户、账号、用户)优先考虑用 HASH 分发策略。分片键字段少的大表，采用 RANGE/LIST 分片策略，例如面向机构编码或地市字段大表采用 LIST 分片策略。

2) 对包含日期的流水表，采用先分片再分区。

3) 每个分片的数据总量不超过 2~3TB，推荐不超过 2TB。

4) 单台服务器的数据总量不超过 6~9TB，推荐不超过 6TB。

5) 单个分片的表记录行总数不超过 1 亿条。

6) 单个分区的表记录行总数不超过 2 千万条。

7) 关联性强的表（比如 JOIN、批量迁移等）尽采用相同的分片策略，例如客户的主档表和流水表，使用相同的分片策略，使得相同客户的多个表数据落在相同的分片上，减少跨分片关联。

8) 避免热点数据的过分集中，如果因为分发策略的原因导致热点数据，可通过增加分发字段的方式，使数据分布进一步细化，热点的判断可通过各个分片的各项指标综合判断。

9) 多级分片，实际使用中出于某些考虑，需要对数据进行复杂的分片。比如集团客户分在某个分片，非集团客户按照客户号 HASH 分片。

10) 先分片再分区，对包含历史表等有日期字段的表，采用先分片再分区的策略。使用年份、月份等字段进行分区设计，将相同分片上的大表按日期将数据分散到多个分区。

3.3.2. 选择恰当的分片主题

为了快速和合理进行表分片设计，在启动分片设计之前，理清各实体之间的关系至关重要。从实体关系中提炼业务的分片主题，比如分为围绕用户（客户、账号、用户）、机构等相同分片主题的表，使用相同的分片

策略，以及相同的分片字段作为分片键。将整个工程中分片主题控制在围绕其中一个主题为主，少量主题为辅助（3~5 个）的规模。

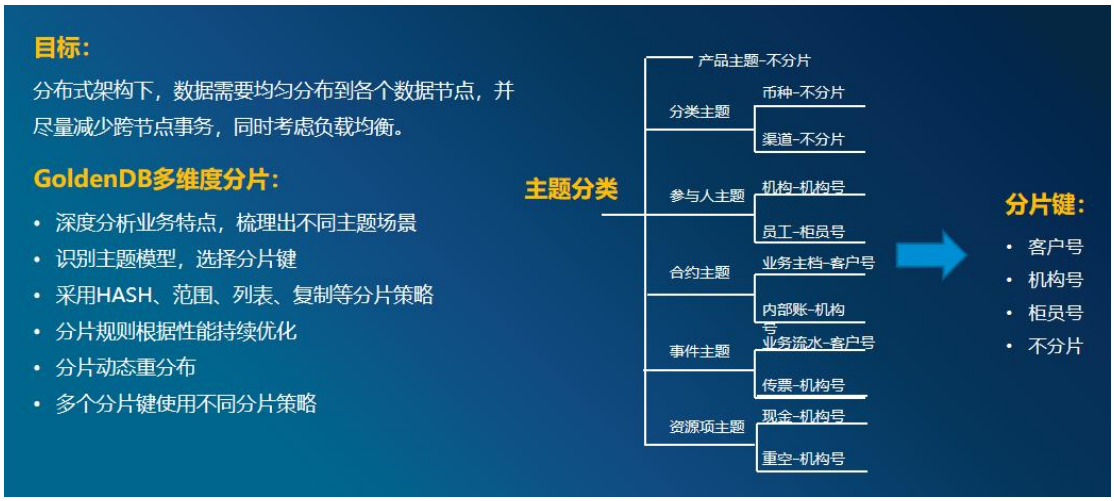


图 13 选择恰当的分片主题

1）分类主题，不分片。

2）参与人主题，机构按按照机构编号进行 RANGE 分片，考虑到机构的规模进行合理的设计，保证各分片数据的均衡。员工按柜员号进行 HASH 或者 RANGE 分片。

3）客户、账号、用户相关的表，可按照客户编号进行 HASH 或者 RANGE 分片。

4）合约主题，业务主档按客户号进行 HASH 或者 RANGE 分片。机构按按照机构编号进行 RANGE 分片。

5）事件主题，业务流水按客户号进行 HASH 或者 RANGE 分片。机构按按照机构编号进行 RANGE 分片。在分片基础上，可对事件发生的时间字段再进行分区。

6) 资源项主题，按按照机构编号进行 RANGE 分片。

3.3.3. 对应用透明的 SQL 开发模型

GoldenDB 分布式架构对于应用开发是透明的，由数据库后台进程自动进行 SQL 解析和分片路由，无需应用关心数据具体存放于哪个分片以及计算分片位置。

下面通过几个典型 SQL 场景说明 GoldenDB 如何自动进行 SQL 解析以及分片路由、多表关联处理，涉及员工信息表和销售信息表，建表 DDL 如下：

```
#员工信息表 employees，根据员工编号自动 hash 分片
Create Table: CREATE TABLE `employees` (
  `id` int NOT NULL,
  `name` varchar(50) DEFAULT NULL,
  `age` int DEFAULT NULL,
  `salary` float DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
 DISTRIBUTED BY HASH(`id`)(g1,g2)

#销售统计表，按照销售记录自动 hash 分片
Create Table: CREATE TABLE `sales` (
  `emp_id` int DEFAULT NULL,
  `sal_recid` int NOT NULL,
```

```
  `sal_date` date DEFAULT NULL,
  `sal_amt` float DEFAULT NULL,
  PRIMARY KEY (`sal_recid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
 DISTRIBUTED BY HASH(`sal_recid`)(g1,g2)
```

1）查询员工编号为 1003 的员工信息

```
mysql> select name,age,salary from employees where id=1003;#employees 使用 id 员工编号作为分片健
+------+------+--------+
| NAME | AGE  | SALARY |
+------+------+--------+
| nola |   32 | 5000 |
+------+------+--------+
1 row in set (0.01 sec)
#通过执行计划可以看到数据库自动进行分片路由，定位到编号 1003 的员工数据在 g1 分片
mysql> explain select name,age,salary from employees where id=1003;
+-------+-------------+-----------+------------+-------+-----------
-------------------------------------------------------------------
----------------------------------+------------+---------+-----------
--------------------+------+----------+--------------------------+
| ID    | SELECT_TYPE | TABLE     | PARTITIONS | TYPE  | POSSIBLE_KEYS
| KEY         | KEY_LEN | REF                                | ROWS | FILTERED
| EXTRA                       |
+-------+-------------+-----------+------------+-------+-----------
```

```
--------------------------------------------------------------------------------
-----------------------------------+------------+--------+-----------
--------------------+------+---------+-------------------------+
    | 10001 | SQLNode      | 1          |            |        |  SELECT
`name`,`age`,`salary`,`testdb`.`employees`.`gtid` as `gtid1` FROM
`testdb`.`employees`  where (`id` = 1003)   | Cluster1,g1 |          |
Parent=NULL,Child=NULL,Next=NULL |       |           | cr,
testdb.employees=hash |
    | 1      | SIMPLE       | employees | NULL        | const | PRIMARY
| PRIMARY      | 4        | const                            | 1     | 100.00
| NULL, g1                   |
    +-------+-------------+----------+-----------+-------+-----------
--------------------------------------------------------------------------------
-----------------------------------+------------+--------+-----------
--------------------+------+---------+-------------------------+
    2 rows in set (0.02 sec)
```

2）SQL2：统计员工基本工资大于 5000 且销售额大于 50000 的员工总数

```
    SELECT count(1)
    FROM employees a
    JOIN sales b ON a.id = b.emp_id
    WHERE a.SALARY > 5000
       AND b.sal_amt > 500000;
    #通过计划拆解可以看到自动对 a 表和 b 表下分片后进行关联聚合
```

```
mysql> explain select count(1) from employees a join sales b on a.id=b.emp_id where a.SALARY > 5000 and b.sal_amt > 500000;
+-------+------------+-------+-----------+-------+--------------------------------------------------------------------
| ID    | SELECT_TYPE | TABLE | PARTITIONS | TYPE  | POSSIBLE_KEYS
+-------+------------+-------+-----------+-------+--------------------------------------------------------------------
| 10001 | JoinNode   | 1     |           |       |
id = 1
child0 = 2 child1 = 3
inner join
table : (nest_last_join)
on expr :  (`testdb`.`b`.`emp_id` = `testdb`.`a`.`id`)
involved tables : testdb.b testdb.a
rowLen() : 0

id = 2, table = b
where expr : (`testdb`.`b`.`sal_amt` > 500000)
sort key : testdb.b.emp_id
involved tables : testdb.b
field list : testdb.b.sal_amt testdb.b.emp_id
rowLen() : 0

id = 3, table = a
where expr : (`testdb`.`a`.`SALARY` > 5000)
sort key : testdb.a.id
involved tables : testdb.a
field list : testdb.a.SALARY testdb.a.id
rowLen() : 0

        | null          |         | Parent=NULL,Child=2,3,Next=NULL |       |          | cr,sort-merge,using where          |
| 10002 | SQLNode       | 2     |           |       |  SELECT `testdb`.`b`.`sal_amt`,`testdb`.`b`.`emp_id` FROM `testdb`.`sale
| 1     | SIMPLE        | b     | NULL      | ALL   | NULL
| 1     | SIMPLE        | b     | NULL      | ALL   | NULL
| 10002 | SQLNode       | 3     |           |       |  SELECT `testdb`.`a`.`SALARY`,`testdb`.`a`.`id` FROM `testdb`.`employees
| 1     | SIMPLE        | a     | NULL      | index | NULL
| 1     | SIMPLE        | a     | NULL      | index | NULL
+-------+------------+-------+-----------+-------+--------------------------------------------------------------------
```

通过上述两个典型场景 SQL 语句可以发现，业务侧在进行 SQL 开发过程中不用关心具体操作记录所处的分片信息以及分片策略，重点关注业务本身逻辑即可，当然，为提升 SQL 执行效率减少不必要的跨分片的操作，这点在库表模型确定分片策略阶段应充分考虑。

3.3.4. 对应用透明的事务控制过程

应用在分布式模式下事务控制和传统集中式数据库一致，由数据库内部保证事务的 ACID 特性，应用无需关心分布式架构内部的事务控制过程。事务控制典型过程如下：

```
function PREPARE_statements()
   if not sysbench.opt.skip_trx then
      PREPARE_begin()
```

```
      PREPARE_commit()
   end

end

function event()
   local src = sysbench.rand.uniform(1, sysbench.opt.table_size)
   local dest = sysbench.rand.uniform(1, sysbench.opt.table_size)

   if not sysbench.opt.skip_trx then
      begin()  //开启事务
   end

   con:query(string.format("select * from zh_fock.account where accno =
'%d'",src))
   con:query(string.format("select * from zh_fock.account where accno =
'%d'",dest))
   //A 账户转出
   con:query(string.format("update zh_fock.account set realtimeremain =
realtimeremain - 0.01 where accno = '%d'",src))
  //B 账户转入
   con:query(string.format("update zh_fock.account set realtimeremain =
realtimeremain + 0.01 where accno = '%d'",dest))
   con:query(string.format("insert into
journal(flowno,flowdate,amount,debitacc,creditacc)
values(null,now(),0.01,'%d','%d')",src,dest))
```

```
   if not sysbench.opt.skip_trx then
      commit() //事务提交，结束事务
   end
end
```

3.4. 提升交易处理性能

3.4.1. 通过索引提升查询性能

索引常被用来优化SQL，提升访问性能。设计索引可以参考如下策略：

1. 选择合适的索引字段。评估出应用中作为查询条件出现比较频繁的字段，在此字段上建立索引，遵循以下的原则：

1）高选择性，选择性是指某列的不同值的数量占该表总数据量的百分比。选择性越高，通过索引查询返回的结果集越少，索引更为高效效。在OLTP 应用系统中，选择性应接近于 100%。

2）数据分布均匀，索引字段中，个别数据值占总数据量的百分率明显比其它数据值占总数据量的百分率高，表明该字段数据值分布不均，容易引起数据库选择错误索引，生成错误的查询执行计划。应该避免在数据值分布不均的字段上建立索引。

2. 选择合适的索引类型。选择单列索引还是组合素引选择，选择方式如下：

1）如果列的选择性非常高(接近于100%)，适合建立单列索引。

2) 根据查询条件,如果单列选择性不高,但组合列选择性较高时,需建立组合索引。

3. 合理使用组合索引。

1) 将具有选择性较高、使用最频繁目经常是等值查询或者 N 的放在组合索引的最左侧。

2) 将具有一定选择性且是范围查询(>、<、>=、<=、BETWEEN...AND)放在组合素引的右侧

3) 排序列必须紧跟在等值条件列的后面能使用该索引避免排序。

4. 对 TEXT、BLOB 和长 VARCHAR，必须使用前缀索引。对于 BLOB、TEXT，或者很长的 VARCHAR 类型的列，为它们的前几个字符建立索引。

5. 合理使用全局索引。全局索引可以有效提升查询条件中无法包含分发键场景的性能。当 where 条件中不包含分发字段时，可将 where 中经常出现的某个或某些字段作为全局索引字段建立全局索引,走全局索引可以快速定位数据所在分片，避免群发。可以通过如下几种方式创建全局索引：

1) 建表时指定 golbal index

```
CREATE TABLE [IF NOT EXISTS] table_name
```

```
col_name column_definition
GLOBAL [UNIQUE] INDEX
    [index_name] [index_type] (index_col_name,...)
    [index_option] ...
```

2) 建表后再创建 golbal index

```
CREATE GLOBAL INDEX index_name
    ON tbl_name (index_col_name,...)
| CREATE GLOBAL UNIQUE INDEX index_name
    ON tbl_name (index_col_name,...)

index_col_name:
    col_name [(length)]
```

3) alter table add global index 新增全局索引

```
ALTER TABLE tbl_name
ADD GLOBAL [UNIQUE] INDEX [index_name]
(index_col_name,...)

index_col_name:
    col_name [(length)]
```

3.4.2. 合理使用分区

GoldenDB 支持分区，且每个分片可以设置不同的分区策略，分区表主要适用于如下场景:按日期访问和归档的表,比如历史表,流水表和明细表。查询时加上时间范围条件效率会非常高,同时对于不需要的历史数据

能很容易的批量删除。使用分区时需要注意：

1）主键、唯一键必须包含分区列，分区表的 primary key 和唯一索引，必须包含用于分区的列。

2）允许按天或者按月分区。

3）联机常用的 SQL 条件中必须要带上分区列作为查询条件。

4）统一对分区进行维护，如需要提前预分配分区，比如按日新建每日的分区，过期的分区需要删余。

5）限制分区数。不建议分区数超过 100。

3.4.3. 提升表关联性能

分布式架构下不同业务表数据根据各自分片策略/主题分布在不同/相同分片上，在处理多表关联时尽量保证：

1）关联字段为各表的分片键。

2）表分片键分片策略尽量保持一致，即相关数据存在于同一个分片，避免跨分片 Join 操作。

3）其它查询列上存在索引，提升数据检索效率。

4）以员工、销售表关联查询为例，查询工资大于 5000，销售额大于 50 万的员工和销售综合和销售日期。两表结构如下：

```
//employees 表按照员工编号 id hash 分片
Create Table: CREATE TABLE `employees` (
  `id` int NOT NULL,
```

```
  `name` varchar(50) DEFAULT NULL,
  `age` int DEFAULT NULL,
  `salary` float DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
 DISTRIBUTED BY HASH(`id`)(g1,g2);
//sales 表初始以销售记录编号 hash 分片
Create Table: CREATE TABLE `sales` (
  `emp_id` int DEFAULT NULL,
  `sal_recid` varchar(50) NOT NULL,--销售记录编号
  `prod_id` varchar(20) NOT NULL,
  `sal_date` date DEFAULT NULL,
  `sal_amt` float DEFAULT NULL,
  PRIMARY KEY (`sal_recid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
 DISTRIBUTED BY HASH(`sal_recid`)(g1,g2)
```

原始查询 SQL 如下：

```
SELECT a.name
    ,b.sal_amt
    ,b.sal_date
FROM employees a
JOIN sales b ON a.id = b.emp_id
WHERE a.SALARY > 5000
    AND b.sal_amt > 500000;
```

执行计划分析，两表关联走的跨分片关联，分别执行 employees 表和 sales 表按条件查询，之后在计算层进行 join 关联操作，得出结果：

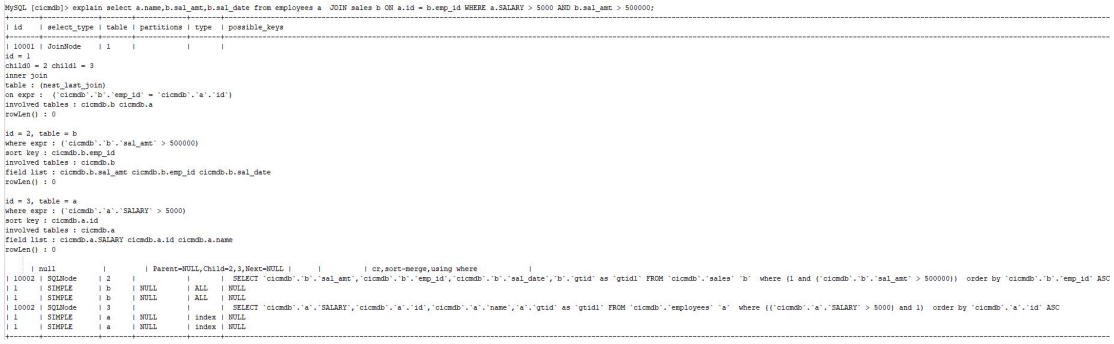


图 14 explain 示例

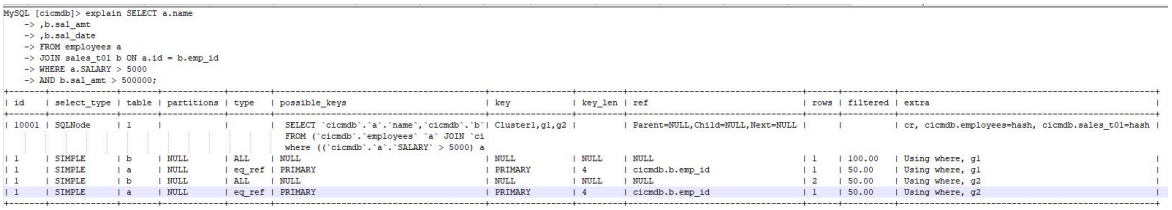
```
MySQL [cicmdb]> SELECT a.name
    -> ,b.sal_amt
    -> ,b.sal_date
    -> FROM employees a
    -> JOIN sales b ON a.id = b.emp_id
    -> WHERE a.SALARY > 5000
    -> AND b.sal_amt > 500000;
+-----------+---------+------------+
| name      | sal_amt | sal_date   |
+-----------+---------+------------+
| johy      |  650000 | 2023-08-01 |
| ciky rose | 1000000 | 2023-08-01 |
+-----------+---------+------------+
2 rows in set (0.50 sec)
```

图 15 实际执行开销示例(耗时较大)

优化思路：调整 sales 表分片策略，如下：

```
//sales 表初始以员工编号 hash 分片
Create Table: CREATE TABLE `sales` (
  `emp_id` int DEFAULT NULL,
  `sal_recid` varchar(50) NOT NULL,--销售记录编号
  `prod_id` varchar(20) NOT NULL,
  `sal_date` date DEFAULT NULL,
  `sal_amt` float DEFAULT NULL,
  PRIMARY KEY (`sal_recid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_bin
 DISTRIBUTED BY HASH(`emp_id`)(g1,g2)
```

再次执行，执行计划如下，此时直接走的分片下压，



```
MySQL [cicmdb]> explain SELECT a.name
    -> ,b.sal_amt
    -> ,b.sal_date
    -> FROM employees a
    -> JOIN sales_t01 b ON a.id = b.emp_id
    -> WHERE a.SALARY > 5000
    -> AND b.sal_amt > 500000;
+-------+-------------+-------+------------+--------+---------------+----------+---------+-----------------------------------+------+----------+-----------------------------------------------------+
| id    | select_type | table | partitions | type   | possible_keys | key      | key_len | ref                               | rows | filtered | extra                                               |
+-------+-------------+-------+------------+--------+---------------+----------+---------+-----------------------------------+------+----------+-----------------------------------------------------+
| 10001 | SQLNode     | 1     |            |        |  SELECT `cicmdb`.`a`.`name`,`cicmdb`.`b` | Cluster1,g1,g2 |         | Parent=NULL,Child=NULL,Next=NULL |      |          | cr, cicmdb.employees=hash, cicmdb.sales_t01=hash |
|       |             |       |            |        |  FROM (`cicmdb`.`employees` `a` JOIN `ci |          |         |                                   |      |          |                                                     |
|       |             |       |            |        |  where ((`cicmdb`.`a`.`SALARY` > 5000) a |          |         |                                   |      |          |                                                     |
| 1     | SIMPLE      | b     | NULL       | ALL    | NULL          | NULL     | NULL    | NULL                              | 1    | 100.00   | Using where, g1                                     |
| 1     | SIMPLE      | a     | NULL       | eq_ref | PRIMARY       | PRIMARY  | 4       | cicmdb.b.emp_id                   | 1    | 50.00    | Using where, g1                                     |
| 1     | SIMPLE      | b     | NULL       | ALL    | NULL          | NULL     | NULL    | NULL                              | 2    | 50.00    | Using where, g2                                     |
| 1     | SIMPLE      | a     | NULL       | eq_ref | PRIMARY       | PRIMARY  | 4       | cicmdb.b.emp_id                   | 1    | 50.00    | Using where, g2                                     |
+-------+-------------+-------+------------+--------+---------------+----------+---------+-----------------------------------+------+----------+-----------------------------------------------------+
```

图 16 explain 示例(优化后)

执行结果如下，性能提升明显

```
MySQL [cicmdb]> SELECT a.name
    -> ,b.sal_amt
    -> ,b.sal_date
    -> FROM employees a
    -> JOIN sales b ON a.id = b.emp_id
    -> WHERE a.SALARY > 5000
    -> AND b.sal_amt > 500000;
+-----------+---------+------------+
| name      | sal_amt | sal_date   |
+-----------+---------+------------+
| johy      |  650000 | 2023-08-01 |
| ciky rose | 1000000 | 2023-08-01 |
+-----------+---------+------------+
2 rows in set (0.00 sec)
```

图 17 实际执行开销示例(优化后)

3.4.4. 使用流式访问快速响应页面请求

执行 SQL 查询数据时，默认从服务器一次取出所有数据放在客户端内存中，返回数据量较大时可能会出现应用响应卡顿。可以采用流数据接收方式，每次只从服务器接收部份数据，直到所有数据处理完毕，避免应用处理迟滞，可以通过如下方式实现流式访问：

1. statement 设置以下属性时，

   setResultSetType(ResultSet.TYPE_FORWARD_ONLY);

2. setFetchSize(Integer.MIN_VALUE);
3. 调用 statement 的 enableStreamingResults 方法，实际 enableStreamingResults 方法内部封装的就是第 1 种方式。
4. 设置连接属性 useCursorFetch=true，statement 以 TYPE_FORWARD_ONLY 打开，再设置 fetch size 参数，表示采用服务器端游标，每次从服务器取 fetch_size 条数据。

示例，url 配置：

```
url=jdbc:goldendb://10.47.161.46:9271?characterEncoding=utf8&useCursorFetch=true&useSSL=false
```

代码中添加红色标记的位置内容，即可以达到预取部分的功能：

```
public void testFetchSize()
{
    PREPAREdStatement ps = null;
    Connection conn = null;
    ResultSet rs = null;
    try {
        conn = DBUtils.getConnection();
        // 预处理添加数据
        ps = conn.PREPAREStatement("select * from test_dev.emp");
        // 设置每次预取的条数
        ps.setFetchSize(100);
        rs = ps.executeQuery();

        while (rs.next()) {// 输出结果
```

```
                System.out.println(rs.getString("job") + "\t" +
rs.getString("ename"));
                }
            } catch (SQLException e) {  e.printStackTrace(); }
            finally {  // 清理资源
                DBUtils.close(rs);
                DBUtils.close(ps);
                DBUtils.close(conn);
            }
        }
```

3.5. 提升复杂查询并发执行性能

GoldenDB AP 引擎引入 MPP 组件，可以有效提升复杂规模计算效率，MPP 充分利用集群中多台服务器的 CPU 以及内存资源，对海量数据进行快速查询分析。计算节点判断业务请求是 OLAP 类 SQL 时，转发给分析型计算节点，由其承担复杂查询、复杂分析的 OLAP 任务，并根据调度策略分发给相应的数据节点。

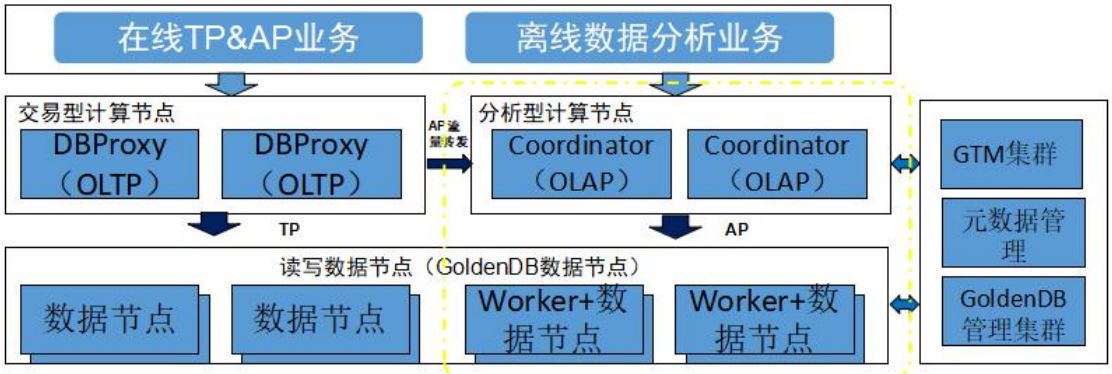


图 18 MPP 组件提升复杂 SQL 执行效率

可以通过如下配置启用 MPP 组件功能

```
#修改计算节点的配置 proxy.ini
conn_instance_for_mpp=8880                    #配置当前计算节点对外开放的服务端口，将对应端口转发到 mpp 运行
default_parallel_processing_mode = 1          #打开并行模式，否则即使设置开放端口也不会使用 mpp
```

3.5.1. 分页设计

Limit 方式下的分页，建议排序字段和 where 条件字段、select_list 字段列表一起建立联合索引，避免大数据量遍历时会有全表扫描或者直接读聚簇索引页，避免计算节点二次排序有大量的 IO、内存、网络开销。推荐的优化方式：

1. 数据按照分页方式进行顺序遍历。每次分页查询记录当前页的最大 id，下一页的查询根据当前的最大 id 作为偏移位置向后取一页数据。该场景有局限性，分页必须是连续的。

2. 将待查询出的目标页的主键先查出来，将主键再带回语句中进行二次查询。这样可以避免在计算节点做二次排序时内存开销太大，减少参与排序的数据行大小也是加快排序速度。如针对如下 SQL:

```
#原始 SQL
Select db.table.col2 from db.table [ where condition ] [ order by xxx] limit 10000,10;
#优化为：
```

```
select id from table [ where condition ] [ order by ] limit 10000,10;
Select cols from table WHERE id in (‘123’,’456’…) ;
#进一步优化:
SELECT a.* FROM a, (select col1,col2 from a order by col1 asc LIMIT
1000000,20 ) b where a.col1=b.col1 and a.col2=b.col2
#其中 col1 和 col2 为联合主键
```

3. LIMIT 分页与缓存结合,先一次性查询多页记录数据或者全部 ID|主键缓存到内存，业务程序实现或通过缓存二次访问数据库来实现分页功能。

3.5.2. 利用分布式隔离级别提升性能

GoldenDB UR（非一致性读）级别下无需判活（判断 GTID 是否在活跃事务列表），可以减少和 GTM 组件的交互以及减少由于活跃 GTID 冲突带来的 GTID 查询等待时间，提升读取性能。使用场景：

1. 对于业务相关的所有表都会集中在某一个分片，与之相关的分布式事务也会集中在该分片。

2. 业务场景涉及查询历史数据，即查询主要涉及历史数据的查询，此部分数据基本不涉及到变更。

3. 某些可接受脏读的业务场景，即部分分片读取到的是全局未提交的数据。示例 SQL：

```
SELECT a.name
    ,b.sal_amt
    ,b.sal_date
```

```
FROM employees a
JOIN sales_t01 b ON a.id = b.emp_id
WHERE a.SALARY > 5000
   AND b.sal_amt > 500000
   AND b.sal_date BETWEEN '2023/01/01'
   AND '2023/1/31' UR;//UR 为分布式脏读隔离级别，读取性能更高
```

3.6. 提升批处理性能

GoldenDB 和传统集中式数据在跑批场景下相比可以充分利用分片的优势，分片间并行执行跑批任务，依托数据均衡分布在各分片，可以把跑批应用程序按功能拆分，拆成多个作业（多个程序）并发处理。分后的几个程序之间没有前后依赖关系，可以并行执行。

1. 为充分利用分布式数据库的特性,请按分片进行批处理,通过 SHOW DISTRIBUTION

   FROM【table _name】table 获取表分片信息后,再通过多个线程使用 STORAGEDB 语法针对多个分片并发处理。

2. 批处理涉及大数据量查询,请通过游标方式先把数据的主键查询到本地,在本地确

   定数据分块的上下边界后,通过使用主键范围作为 WHERE 条件分批次查询数据。

   参考示例,跑批主体查询 SQL 如下：

```
select distinct iaa.*
from CIF_CLIENT iaa, MB_ACCT ia
where iaa.CLIENT_NO = ia.CLIENT_NO
and ia.ACCT_STATUS != 'C'
and ia.INTERNAL_KEY <= 25625894
and ia.main_bal_flag = 'Y'
and ia.int_ind = 'Y'
and (ia.source_module !='CL'
OR (ia.source_module = 'CL'
AND ia.lead_acct_flag != 'Y'))
表:
cif_client 8087184 rows,
mb_acct 22285821 rows
```

为充分发挥分布式模式分片并行处理能力，优化如下：

1）cif_client 和 mb_acct 使用 CLIENT_NO 做分片键(使得两表 JOIN 能下压执行)。

2）进一步分析 mb_acct 表和 cif_client 哪张表是主表,以主表的主键作为分片键，将主表的主键字段作为冗余增加到从表，并以增加的冗余字段作为从表的分片键。

3）优化索引，CLIENT_NO，ACCT_STATUS，INTERNAL_KEY，main_bal_flag，int_ind，source_module 根据情况创建本地索引。

应用侧优化如下：

1）查询表涉及的分片数：SHOW DISTRIBUTION FROM cif_client；返

回 g1，g2，g3 三个分片。

2) 每个分片(g1,g2,g3)对应启动一个 Pod(共 3 个)；

3) Pod 内主控进程查询该分片 mb_acct 表的主键范围：select pk from mb_acct；按主键范围分成 64 段(每段 12 万 CLIENT 记录)；

4) 启动 64 个 work 进程，每个 work 进程处理一段记录：select ... from CIF_CLIENT iaa, MB_ACCT ia where client_no between 分段 i;

5) 开启流式处理,数据库扫描 2000 行就返回结果给批量应用（PREPAREdStatement.setFetchSize(2000)；）；

6) work 进程查询出 2000 行记录时，再做进一步计算、更新数据库操作，commit 事务并记录批量进度(断点重试位置)，直到结束。

3.7. 通过读写分离提升查询性能

读写分离提供了一种将特定的读语句按照预配置的分发规则发送到数据节点分片中的主机或者备机，主要是备机，以此减少对主机的访问，同时提升备机的资源使用率。

针对部分实时性要求不高的查询,可以通过读写分离分发到备机查询，如：历史明细、参数查询。支持实例级和语句级：

1) 在连接实例上配置，可选择的配置类型有：不开启读写分离（默认）、本地同城策略、异地策略，其中本地同城策略需要配置权重；

2) 语句级别的读写分离策略,通过在 sql 语句中携带 hint 实现。hint

包括 READMASTER、READBALANCE、READSLAVE。

注意：主备 DB 时延超过阈值时不进行读写分离。

GoldenDB 读写分离支持灵活配置读写分离的权重，可以按照本地同城/异地开启权重配置。

通过 Insight 平台配置读写分离，主要步骤如下：

1）租户管理页面->租户->服务端口->点“编辑”弹出服务端口页面

图 19 设置服务端口参数

2）设置读写分离策略，编辑服务端口页面并点“确定”使之生效。

图 20 设置服务端口的读写分离策略

3.8. 数据导入导出

3.8.1. 使用数据导入导出工具

GoldenDB 支持分布式数据库批量数据导入，实现异构数据库之间的数据迁移。如将 oracle 数据库数据导出为数据文件，可使用 GoldenDB 工具（LoadServer）导入到 GoldenDB 数据库中。LoadServer 用户也支持批量导入应用端生成的数据等。LoadServer 还作为 GoldenDB 数据逻辑备份，下游用户应用进行数据分析、统计、处理等场景。

导入支持导入全部字段、部分字段、set 子句、when 子句，同时支持空列以及数据切片提升分片数据并行导入能力，如下为数据导入带 when 子句示例：

```
dbtool -loadserver -type="in" -clusterid=6 -sql="load data infile 'loadout.data' into table load_manual.t0101_in fields terminated by ',' optionally enclosed by '\"' lines terminated by '\n' ( id,hotelname,year) when year>2013 ;" -user="userA" -password="passA"
```

相应导出功能支持导出全部、部分字段、where 条件，支持导出按行数和文件数进行切割以及指定分片导出,如下为带有 where 条件的导出示例：

```
dbtool -loadserver -type="out" -clusterid=2 -sql="select id,hotelname,year ,month ,day , rent from load_manual.normal where id<10 into outfile 'normal.data' fields terminated by ',' lines terminated by '\n';" -user="userA" -password="passA"
```

3.8.2. 向下游数据库同步数据

SLOTH2.0 是 GoldenDB 自研的数据库迁移工具集，包含迁移评估、数据同步和采集回放 3 大工具，其中数据同步工具可以提供同构/异构数据同步、支持全量和增量的同步模式以及实时数据一致性校验，目前已支持的同构和异构数据库包括：

- ORACLE
- MySQL
- TDSQL
- GoldenDB
- DB2
- KAFKA

3.9. 数据库开发规范化

制定一套应用开发遵循的开发规范，有利于避免重复触及数据库使用常见问题，提高可阅读性和问题定位速度。

3.9.1. 统一命名规范

数据库命名规范。数据库名应尽可能和所服务的业务模块名一致。不同的项目使用不同的数据库名，不允许不同的项目使用相同的数据库名。数据库名必须简洁，建议不要超长。

表命名规范。同一个租户的表使用相同的前缀，表名称需要表达该表

数据的业务含义。原则上命名以词根组成为准，超长可以只选择部分关键词根表示。所有表的表名以租户开头，不同类型的表命名规范:

1） 历史表以”_hist”作为后缀。

2） 明细表以”_detail”或者”_dtl”作为后缀。

3） 参数表以”_param”作为后缀。

4） 关联关系表以”_rel”作为后缀。

5） 映射表以”_map”作为后缀。

6） 临时表以”_tmp"作为后缀。

7） 中间表以”_mid”作为后缀。

8） 程序需要访问的备份表”_copy”作为后缀。

9） 程序不会访问的备份表头”_backup"+日期或者”_bak” +日期作为后级。

10） 对于联机和批量表结构类似,使用用途一样时。联机用的表,在第二段以”_online_”.

批量用的表,第二段以”_batch_”。

列命名规范。列命令必须符合单词命名规范。常用命名的约定:

1） 标识符字段:以”_id”后缀,如：user_id 表示标识符。

2） 序号列字段:以”_no”后缀,如:trx_seq_no 表示交易序号。

3） 代码字段：以”_code”后缀,如:dist_code 表示地区代码。

4） 日期字段(类型是 date,只有年月):以”_date”作为后缀。

如:open_date 表示开户日期。

5） 时间字段(类型是 time,只有时分秒):以”_time” 作为后缀。如：close_time 表标关门间。

6） 日期时间字段(类型是 datetime，包含年月日时分秒):以”_dt”或者”_datetime” 作为后缀。如：register_dt 表示注册时间。

7） 时间戳字段(类型是 datetime(6),包含年月日时分秒毫秒微妙):以”_ts”作为后缀。如:create_ts 表示创建时间戳。

8） 布尔值字段:以”_flag” 作为后缀。如:表示逻辑删除意义的字段可命名为 delete_flag。

索引命名规范。索引命令的约定:

1） 唯一素引以“udx_”表名方式命名。如果有多个唯一索引,使用”udx_表名 N” 来命名，N 是数字序号,从 1 开始,每次加 1。

2） 辅助(二级)索引以 “idx_表名_N”命名,从 1 开始,每次加 1。

3） 全局索引以 “gdx_表名_N”命名，N 表示数字。

3.9.2. 制定 SQL 开发规范

为提高应用侧开发人员数据库的开发质量、规范 GoldenDB 数据库设计、开发和管理策略，提高数据库的性能和稳定性，总结整理了一些常规的 GoldenDB 开发规范。

3.9.3. 合并插入

如果同时执行大量的插入,请使用多个值的 INSERT 语句。这比使用分开 INSERT 语块 一般情况下批量插入效率有几倍的差别，此种方式减少 SQL 语句解析的操作,只需要解析一次就能进行数据的插入操作，同时可以有效减少网络传输的 IO。优化示例：

```
insert into tablename values(1.2),(2,3),(3 4)
```

3.9.4. 禁止使用大事务

对大表进行更新,要避免大事务,避免主从同步发生大的延迟,触发降水位，影响联机交易，因此建议单个事务务不要超过 1G（具体可根据实际网络情况调整）。

3.9.5. 联机交易的 SQL 禁止使用全表扫描

禁止原因：由于联机交易,一般处理的记录数很少,很适合通过索引来优化联机交易。当执行计划是全表扫描,当表数据量很大时,无索引查询导致高开销的数据库磁盘 IO。除非该表是系统配置表,数据量很小,记录行数小于 1 千行。优化方法：增加基于条件的索引。

3.9.6. 只取出所需要的列,禁止使用 SELECT *

使用查询语句是要避免使用 select *,而是应该指定具体需要获取的字段。如果使用 select *取出全部列,缺点如下:会带来更多的 CPU、IO、

内存和网络消耗;需要查询系统表得到列信息;优化器无法使用索引覆盖;表结构变更可能会对程序带来的影响。规范的写法:

```
select c1,c2,c3 from t1;
```

不规范的写法:

```
select * from t1;
```

3.9.7. 联机交易的 SQL 禁止使用全索引扫描

禁止原因:由于联机交易,一般处理的记录数很少,索引很适合优化联机交易。当执行计划是全索引扫描,当表数据量很大时,全索引扫描导致高开销的数据库磁盘 IO。除非该表是系统配置表,数据量很小,记录行数小于 1000 行。优化方案:建立高效索引。

3.9.8. 禁止对索引列使用函数

原 SQL 示例,表 card_detail 有索引(ac,update_time),update_time 为 datetime 数据类型,有如下查询:

```
select ac,amt from card_detail where ac=? and date(update_time)= ‘2010-01-01’
```

优化 SQL 示例,SQL 修改为

```
select ac,amt
from card_detail
Where ac=?
and update_time>= ‘2010-01 -01 00:00:00’ 、
and update_time< '2010-01-02 00:00:00’
```

3.9.9. 禁止使用前缀进行模糊前缀查询

在使用 LIKE 关键字进行查询时，如果匹配字符串的第一个字符为”%”,不能匹配索引。只有”%”不在第一个位置,才能匹配索引。原 SQL 示例，应避免使用：

```
select id,val from table where val like ‘%name’;
```

优化 SOL 示例，可以使用%模糊后缀查询如：

```
select id,val from table where val like ‘name%’;
```

3.9.10. 排序和分组

排序和分组都会消耗数据库资源,使用不当可能造成性能问题,其注意事项如下:1）减少使用 order by、group by 和 distinct ，和业务沟通是否必须排序，减少排序操作，或者将排序放到程序中处理；2）order by、group by、distinct 这些 SQL 尽量利用索引直接检索出排序好的数据。例如 where a=1 order by b,可以利用(a,b)索引来避免排序；3）包含了 order by、group by、 distinct 这些查询的语句，where 条件中过滤的结果集控制不要超过 1 万条，否则 SQL 执行效率会降低。4）减少跨分片 order by、group by 和 distinct。

3.9.11. 禁止 2 个大表的大结果低效关联

超过 100 万条记录录的表算作大表。

禁止原因：两个大表关联 JOIN 时,需要将每个大表的数据从数据节点

获取后,在计算节点进一步处理。当缺少关联条件判断或关联性不强时,需要将所有大表数据都提取到计算节点,网络容易成为瓶颈,计算节点积压大量中间数据,堵塞他连接请求。除非两表分片键相同且有分片键等值的查询条件,此时 JOIN 可以下沉到多个数据分片上进行本地计算。

优化方法：通过合理设计,将大表 JOIN 下沉到数据节点,避免计算节点阻塞通道。如将两表设计为相同分片规则,且明确查询条件是两表基于相同分片键的等值查询。

3.9.12. 联机交易和批处理程序不允许大结果集

禁止原因：大结果集查询容易对磁盘 IO、网络造成冲击，影响高并发事务交易,阻塞其他连接请求。

优化方法：根据业务场景在应用层控制最大结果集，如分页列表最多显示 2000 行。分批多次查询,每次只查询指定记录行数。

原 SQL 示例

```
Select col1,col2 from tab01;
```

优化 SQL 示例

```
//分多次查询
SELECT col1,col2
FROM tab01
WHERE col3_date > '2023-3-9'
AND col3_date <= '2023-3-10';
```

4. 数据库迁移

4.1. 数据迁移调研评估

4.1.1. 兼容性评估

GoldenDB 提供一套兼容性采集分析工具集，包含迁移评估，并行迁移，以及性能评估三大核心任务，以只读的方式部署，持续不断的进行 SQL 采集,将采集到的全量 SQL 在兼容系统进行评估分析生成兼容度报告。

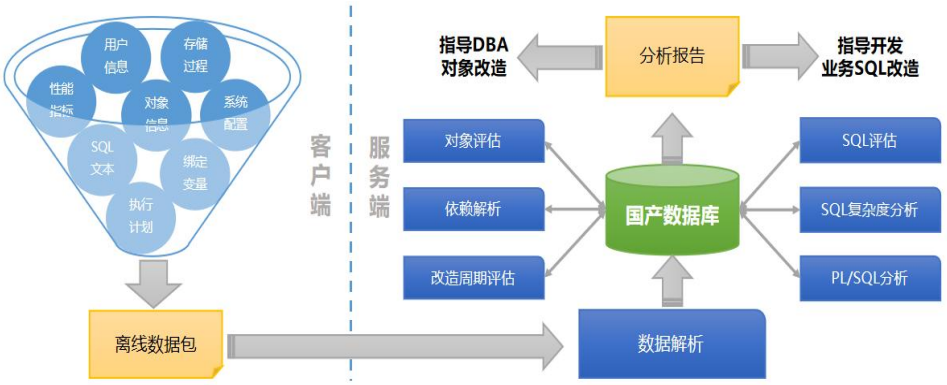


图 21 CACtool 迁移评估功能结构

GoldenDB CACtool（迁移评估工具）通过对源库进行信息采集、分析，预估迁移的可行性、GoldenDB 对源库的兼容情况以及迁移到 GoldenDB 的代价成本（包含兼容程度/改造成本、组网代价等）。

CACtool 通过对源库采集 ddl/dml/dql/各类统计信息等进行分析，支持对表结构(字段类型)、序列、函数、存储过程、同义词以及 SQL 语句进行兼容性分析，并根据内置规则自动进行 DDL 转换，通过对 SQL 分析结果和采集的表信息进行表画像并根据表画像和各类内置规则计算分发方式进行 SQL 兼容性评估，最终生成兼容性分析报告，整个报告包含：

1）源库采集到的各类统计信息、ddl/dml/dql 等

2）转换后的分布式 ddl（分析推荐）

3）兼容性评估结果

4）整体分析报告和迁移建议

5）过程中的各种辅助日志

6）数据库对象迁移包括：数据库用户和权限、数据库表、分区、索引、视图、存储过程、包、函数、触发器、序列、DBLINK 等数据库对象迁移。

4.1.2. 数据迁移调研

为确保正常数据迁移，数据迁移前一定要做调研：

一是明确数据迁移要求，如迁移方案（离线、在线等）、割接窗口期大小、源库数据量大小、是否需要反向同步、数据库字符集（建议与老系统一致，除非不支持）是否需要调整等等；

二是通过调查分析了解源库的真实信息：

1）调查并验证源库的硬件资源情况、业务承载情况，以及所支撑的并发数，波峰波谷情况、日志归档配置及归档空间是否足够等，为评估迁移同步计划以及割接窗口期做准备。

2）源数据库到目标数据库间的网络质量评估。确保端到端的万兆网络，并实际测试验证。如网络质量差（千兆或者更差），需考虑延长数据

同步时间并降低并发量。

3）与客户及业务厂家确认，哪些表仅需同步表结构，哪些表的表结构和数据都无需迁移（如零时表，tmp 类表），哪些表即迁移表结构，又迁移表数据。

4）调查并验证源库总数据大小、物理表数量和各表总行数；

5）源库是否有表压缩，要保持迁移前和迁移后保证相同的压缩属性。

6）源库各表是否有伪列，数据迁移配置时需去除。

7）源库关键表静态分析。需对超过 20G 的表、超过一千万行的表、超过 150 字段以上宽表、含有 LOB 字段的表专题分析，为配置迁移调度策略做准备，超大型表可提前整表迁移或者不变分区迁移。

8）源库关键表动态分析。需对源库的热点表、周期性 DDL 操作的表及分区（turncate、Drop、Create、Move）进行调查分析，为提升数据迁移效率，这些表可能需要特殊处理，比如 turncate 表/分区，可以仅在割接窗口期进行全量迁移即可。

9）统计源库无主键、无唯一索引的表以及各表行数，为提升数据同步速度，可能需对这些表增加索引。

10）对于割接窗口期很短的应用场景，需对关键表进行专题分析，如针对更新或插入频繁的热点大表是否可以分阶段全量迁，小的热点更新表可以在割接窗口期全量迁。

11）调查是否含有 GoldenDB 不支持的数据类型，并与客户沟通是否

可以不迁移或者数据类型变更。

12) 是否有含中文字段的表（多是用于运维的非关键表，可废弃），向客户建议不迁移；

三是迁移服务器集群资源评估。很多的核心业务系统需要在短时间内完成大批量的数据迁移，对服务器规格有很高的要求，因此对这类数据迁移场景建议采用高配服务器（4 路 CPU、512G 内存以上、万兆网卡等），且至少 3 台（可根据具体要求增加）。如数据迁移量不大，且迁移窗口长，也可以选择低配的服务器，或者与 GoldenDB 集群合设。

4.2. 迁移方案确定

基于前期的调研信息，确定 GoldenDB 数据库迁移方案，方案中包括：

4.2.1. 选择整体迁移方案

方案一、基于全量同步的割接方案，无需反向同步。

对于允许离线迁移的业务系统，建议采用全量迁移方案。在全量割接期间，Oracle 源库需停止数据更新，GoldenDB 迁移工具 Sloth 采用全量迁移模式，进行全量数据采集、回放以及全行比对，最后阶段采取各表 count 计算，关键表进行 sum 计算的核验方式，确保源库目标库数据完全一致。

方案二、基于全量+增量+比对+反向同步方案。针对需要反向同步到源库来进行兜底的应用场景，建议采用此方案。

4.2.2. 明确数据迁移时长

数据迁移总时长要保证至少两次全量+增量迁移+追平所需的时间，在此时间基础上越短越好，一是缩短对业务的影响，二是迁移时间过长会因各类异常事件可能导致迁移计划失败。

针对金融联机交易系统停机窗口十分紧张，因此，实际迁移均采用全量+增量的方式进行数据迁移，全量迁移通过对历史数据进行迁移，可以提前以离线（备机）的方式在不影响联机交易的情况下进行迁移，之后通过数据同步工具搭建正向同步通道，实时增量复制源端数据并同步至目标库，期间对联机交易无影响，待新系统割接上线当晚，完成最后增量数据同步后，经过必要的数据校验环境，既可以在尽量小的停机窗口内进行新旧系统数据库的切换，从而完成系统系统的上线。

4.2.3. 确定相关软件配合方案

1）申请足够的数据迁移工具权限，包括不限于端到端的网络权限以及源端数据库的访问权限。

2）源端生产库的配置修改申请。由于 GoldenDB 迁移工具基于归档日志进行增量数据同步，为了更好的实现数据迁移，需修改源库的归档周期到合适的范围，并设置归档最大保存时间（归档保存时间宜大于总迁移时长。

3）数据库数据迁移期间需要申请客户 DBA 配合割接，需清晰的描述：

时间、步骤、动作内容，以及注意事项。比如停止生产库日常运维操作（表空间和表或表分区等 DDL 操作）；何时启停 ADG 备库；是停 ADG 同步还是停止回放；是否需要检查 SCN 或者查杀未提交的长事务等等；割接窗口期内，待应用停止数据变更后，锁定生产库用户等等。

4）迁移过程中生产系统的配合：A）迁移期间应避免 DDL 操作，禁止不刷 redo 日志的数据导入操作，停止 rename 表等操作；B）停止非必须的大事务、长事务等等。

4.2.4. 明确数据校验策略

全量和增量阶段采取全行一致性比对策略；反向同步阶段，由于业务无法停机，无法进行准确比对，多采用割接前进行多次反向同步验证（停库以便保证数据静止，再进行一致性比对）；也可以在生产状态下，通过停止备库的方式进行一致性比对验证。

4.2.5. 明确数据迁移步骤

为避免数据迁移失败，割接方案应细化到每个操作步骤的前置条件、操作指令、预期结果、操作时间等。

4.2.6. 明确 Sloth 数据迁移调度策略

为确保全量数据迁移的速度，建议将全量迁移的任务按单分片复制表、

多分片复制表、以及水平分片表分开，单分片复制表的任务数是迁移服务器数的最小公倍数的 N 倍（N 大于等于 3），每个小任务的数据同步表都在同一个分片内，确保迁移任务均衡。

4.2.7. 迁移回退策略

预先明确哪种场景必须回退，如关键数据表或数据库对象不一致或者不可用、关键业务测试验证不通过、性能不满足业务需要等等。对于非严重故障，只要提前有可靠的应急方案，可以不回退。

4.3. 数据库迁移过程

从异构数据库（Oracle、DB2、Mysql 等）到 GoldenDB 数据库迁移工作包括:数据库对象迁移、数据库语法改造、数据迁移、数据一致性校验。数据迁移包括全量数据迁移和增量数据迁移，主要迁移或同步表数据。

GoldenDB 使用 SLOTH 平台工具支持各类数据库到 GoldenDB 的迁移，工具支持配置全量和增量任务进行迁移，同时支持数据校验，包括全量离线数据校验和实时在线校验。完整的数据迁移过程主要包含如下四个步骤：

步骤 1：全量数据迁移。

步骤 2：增量数据同步。

步骤 3：引流切换。

步骤 4：回流同步。

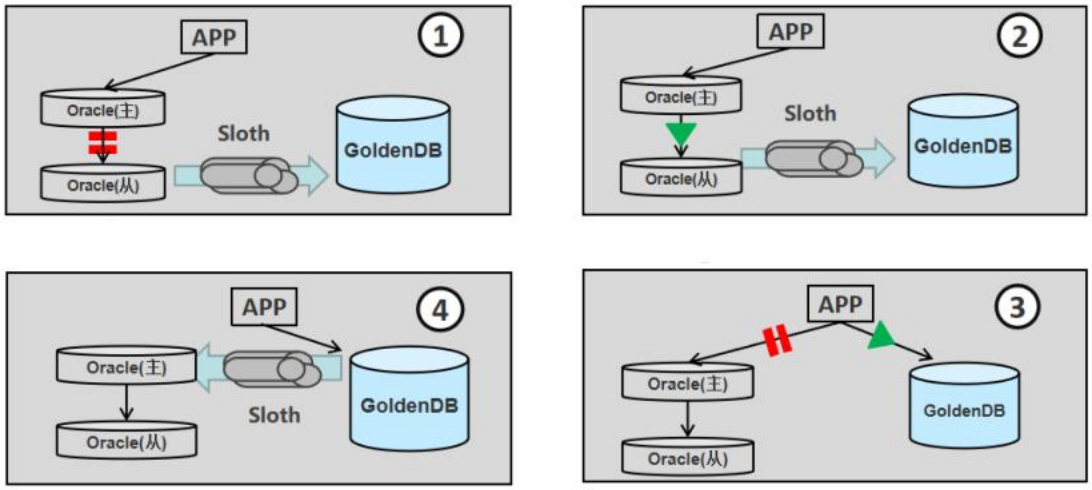


图 22 数据迁移主要步骤

4.3.1. 数据库正向同步

1. 全量迁移流程

全量迁移持续时间和存量数据量成正比，分为迁移和核验两个阶段。全量数据迁移步骤：

1）在 ADG 备库停止同步后，获得生产快照，基于 JDBC 接口进行全量数据迁移，迁移期间主机同步一直处于关闭状态。

2）同步比对机制，比对功能与全量迁移同步启动，期间会比对很多次，当数据量达到阈值或则等待的时间达到阈值（都可以配置），则触发一个轮次的比对过程。当次参与比对的数据包括回放端新同步完成的数据，同时还包括上一轮次比对不上的差异化数据。

3）全量迁移完成后，若有比对不一致的文件，则根据实际情况进行数据修正。

4）全量迁移完成后，根据分片映射规则，生成分片映射表数据。

2. 增量迁移流程

增量迁移在全量迁移完成，确认移植数据与主机镜像数据 100%一致后再进行，主要完成数据迁移期间累积日志的数据追平和生产实时产生的日志跟帐。增量迁移包括所有迁移范围的文件。增量数据迁移步骤：

1）增量迁移，在主机镜像启动同步时，通过下载主机日志并将日志转码、根据转码日志操作符将日志转换为 INSERT、UPDATE、DELETE 等 GoldenDB 可执行 SQL 语句，然后将这些已转换完成的 SQL 语句并发执行，达到主机与 GoldenDB 数据库保持一致的目的。

2）增量比对，比对功能与迁移迁移同步启动，期间会比对很多次，当数据量达到阈值或则等待的时间达到阈值（都可以配置），则触发一个轮次的比对过程。当次参与比对的数据包括回放端新同步完成的数据，同时还包括上一轮次比对不上的差异化数据

3）日志操作条目数比对，在增量迁移的同时每隔 30 分钟进行一次日志条目数比对，主要比对在间隔时间内增量迁移转码操作数（每张表的 INSERT UPDATE DELETE 数量）是否与主机生产上这段时间产生操作数相同，另外，还比较转可执行的 SQL 语句数量是否与操作数匹配。

3. 数据库比对

Sloth 支持在全量迁移、增量迁移、全量+增量迁移任务中，都有对应的采集组件、回放组件和比对组件，也即比对组件是与同步任务绑定的。如下图：

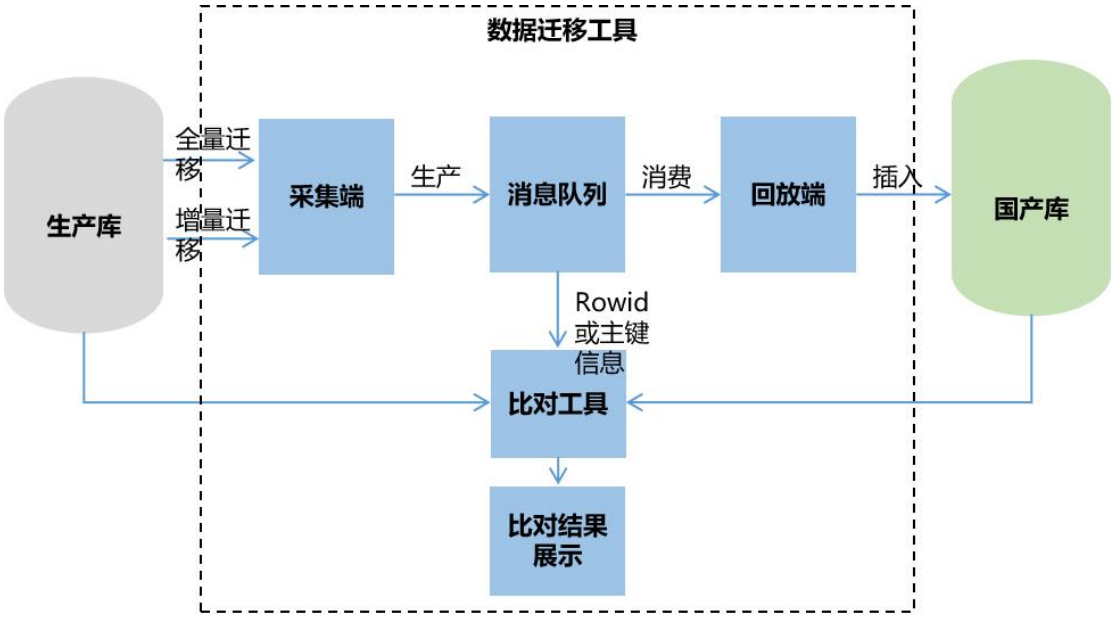


图 23 SLOTH 整体架构

在全量迁移期间进行的数据比对，对应的就是全量的数据比对；在增量迁移期间进行的数据比对，就是增量数据比对。

无论是全量数据比对，还是增量数据比对，都是跟同步任务一起运行，期间会比对很多轮次，当数据量达到阈值或则等待的时间达到阈值（都可以配置），则触发一个轮次的比对过程。当次参与比对的数据包括回放端新同步完成的数据，同时还包括上一轮次比对不上的差异化数据。

对于不匹配的数据量如果很小，可以从界面下发“修复”操作进行一键修复。若不匹配的数据量很大，则说明迁移过程可能存在某些问题，需

要重点关注同步工具的迁移过程，进行多方会诊。

4.3.2. 数据库割接

GoldenDB 完成全量或者全量+增量同步后，可以择机进行数据库切换，将应用访问数据库连接切换到 GoldenDB，对外提供服务。具体步骤如下：

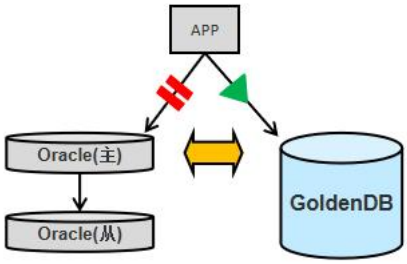


图 24 引流切换过程

数据比对。在切换前可以对数据进行比对验证，确保切换时刻两边的数据一致。如果是全量迁移，对全量数据进行校验比对后，再执行引流切换。如果是增量同步，应用请求停止向 Oracle 写入数据，等 GoldenDB 追完日志并全部回放完成，并进行增量数据比对校验，再执行引流切换。

修改应用连接数据库地址，指向 GoldenDB 数据库。

放开应用前端请求流量，开始对外提供服务。

4.3.3. 回流同步

数据库请求引流后，正式完成数据库迁移切换；作为逃生通道，原

Oracle 数据库环境继续保留一段时间，作为备份数据库，反向接入到 GoldenDB 数据库集群继续追增量数据。反向增量使用 GoldenDB Sloth 工具实现。

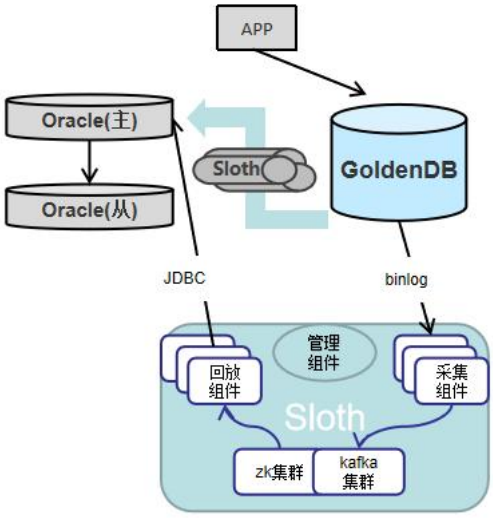


图 25 回流同步

Sloth 增量同步工具由日志采集端(面向源端数据库)、Kafka 队列和 SQL 回放端(面向目标数据库)，源端数据采集节点模拟 GoldenDB 从节点，接入源端 GoldenDB，获取 binlog 数据，使用中间格式存入 Kafka 队列，目的端从 Kafka 队列中取出数据，还原为 SQL 回放到 Oracle 主节点。

4.4. 迁移测试与演练

4.4.1. 迁移环境部署

确认部署环境正确：服务器（用于迁移工具及 GoldenDB 部署的）安装部署完成、网络正常；

确认所有服务器已经实施硬件加速。如配置睿频并取消节能模式，开启 raid 卡缓存，并对迁移实例采取绑核操作。

确认 SLOTH 和 GoldenDB 所有的配置参数正确；

确认 GoldenDB 所在的 DDL 是 Oracle 生产库的最新 DDL。

4.4.2. 迁移模拟演练

迁移演练的主要目的：

验证割接计划的实施步骤是否准确；

验证各阶段的实施细节和各步骤实施时间是否准确；

通过多次模拟割接找出最优的实施方案并缩短割接时间；

通过模拟割接来理顺各参与单位的协同和配合

总结模拟割接演练过程中碰到的异常情况，进行改进和补充应急方案

因此，建议在正式割接前，至少进行 3 次带业务的模拟割接，从模拟割接中总结经验，并适时的进行方案优化调整。 通用的模拟割接演练步骤如下：（T 日代表业务割接到 GoldenDB 当晚）

T-1 日完成环境准备和配置检查

T 日开始全量迁移

T 日完成全量迁移、全量比对、全量阶段一致性校验、差异化数据修复

T 日开始增量同步

T 日增量追平源端、启动增量比对

T 日完成增量比对、增量阶段一致性校验、差异化数据修复

T 日实施数据库割接

T 日开始回流同步

T 日完成回流同步、回流比对

T 日完成回切演练

4.5. 正式割接迁移

经过多次的模拟割接验证，割接方案经过多方评审后，严格按照割接方案实施，当出现异常情况，请根据提前准备的应急方案进行处理，确保割接顺利进行。

注意事项：数据迁移过程中，GoldenDB 应禁用触发器、外键、JOB 等对象，并禁止外部应用写入，避免数据不一致。

5. GoldenDB 日常运维

GoldenDB 具有完备的日常运维体系，方便用户从不同角度多方位地观察整个分布式数据库系统的运行情况。GoldenDB 数据库日常运维包括对系统的监控管理、诊断运行过程中出现的各类故障、检查数据库运行状态是否正常、对数据库产生的告警信息进行及时上送和处理，以及分析系统中存在的性能问题，确保数据库稳定高效地对外提供服务。

5.1. 集群监控

在 insight 运维平台上，按集群维度对数据库运行情况进行监控，包括集群下各组件的运行状态、性能指标数据、资源使用情况、分片的主备复制时延情况、库表详细统计信息等。参考步骤：

➢ 选择菜单[租户管理]，进入租户列表界面，可看到各个集群的名称、状态等信息，可看到集群的基础信息、配置信息和部署信息；可直观地查看一个分片下主备节点之间的赋值时延数据；

图 26 GoldenDB 租户管理

➢ 拓扑图界面，可以直接看到各个组件运行情况，包括主备复制时延是否超阈值、运行状态是否异常、告警数量等。

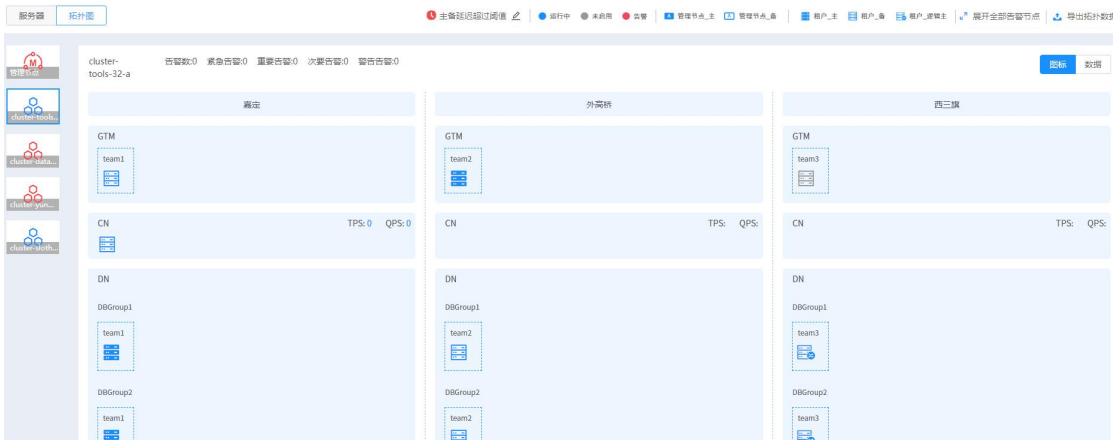


图 27 GoldenDB 集群拓扑图

➢ 性能页面显示了该进程使用的相关资源信息，鼠标移动到图表内，可以显示具体的数据信息

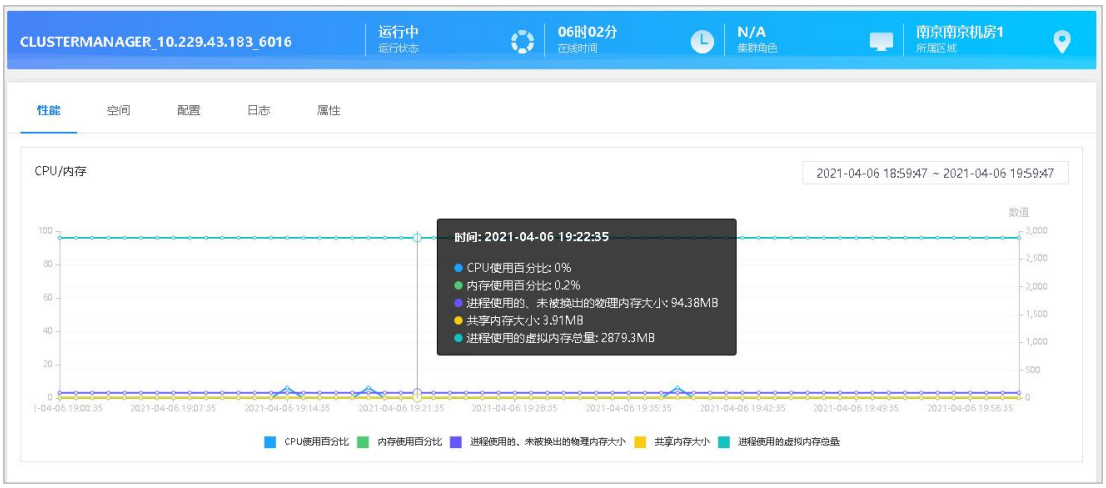


图 28 GoldenDB 性能监控

5.2. 告警监控

在 GoldenDB 运行过程中，管理节点和相应的代理组件会对各个组件进行告警监控，将产生的告警汇总到 insight 运维平台进行展现，运维平台提供对告警的管理操作功能，并且提供告警订阅功能，将告警信息上送至外部订阅者。

➢ 选择菜单[告警管理→实时观察]，进入实时告警查看界面，查看当前系统所存在的所有实时告警，并对实时告警进行相应管理操作

图 29 GoldenDB 告警管理

➢ 选择菜单[告警管理→历史查询]，进入历史告警查询界面

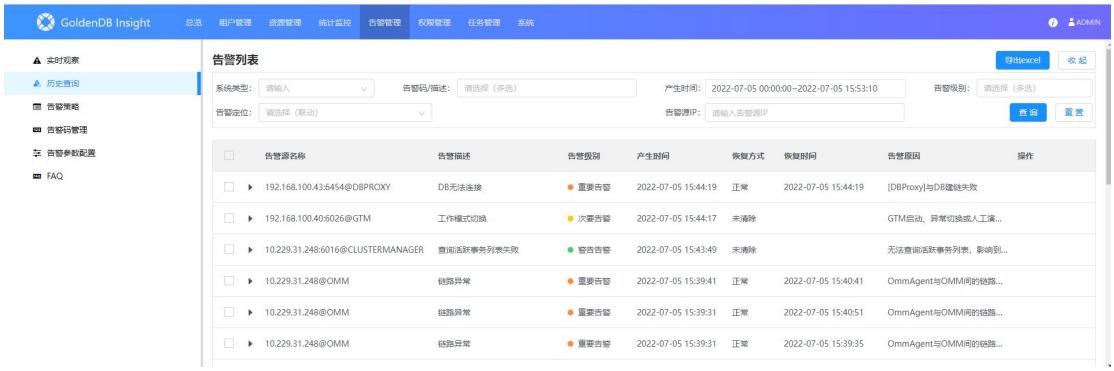


图 30 GoldenDB 告警历史

- 设置查询条件，单击**查询**按钮，显示出根据条件筛选后的告警记录
- 单击**导出 excel** 按钮，可以将查询的告警信息保存到本地 Excel 中

➢ 可设置告警发送策略，在发生告警时，将按照策略发送相关信息给目标；可设置告警屏蔽策略，新发生告警将按照屏蔽策略屏蔽

图 31 GoldenDB 告警策略设置

- 在告警策略页面，单击新增按钮，在弹出的对话框中新增方式选择发送策略
- 在告警策略页面，单击新增按钮，在弹出的对话框中新增方式选择屏蔽策略

图 32 GoldenDB 新增告警策略

➢ 告警码管理。告警码是每个告警的唯一标识，每个告警都有告警级别，用于标识该告警的严重程度。单击查询按钮，可设置查询条件进行告警码的查询，并且支持调整告警码的告警级别。同时，告警原因码中的告警描述信息可由用户定制化编辑修改。

➢ 可在该页面修改告警阈值、配置告警订阅、设置告警静默。

5.3. 节点扩容

通过 insight 运维平台可以便捷地对集群进行扩容缩容操作，下面以 DN、CN 组件的扩缩容为例介绍主要的操作步骤：

➢ DN 节点扩容。选择菜单[租户管理]，在左侧实例树中，展开租户实例

下的数据节点，单击待管理数据节点的分片，进入数据节点管理界面

图 33 通过租户管理页面进行节点扩容

- 可以选择扩容模板克隆或自定义扩容安装模式。
- 扩容过程中可通过进度条查看分片安装进度。

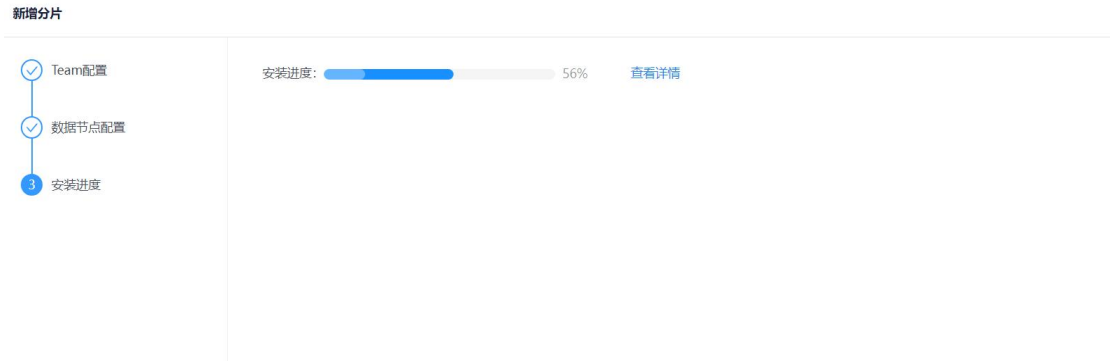


图 34 GoldenDB 扩容进度条

5.4. 全局一致的备份恢复

通过 insihgt 运维平台可对租户下的数据进行备份恢复任务的配置，查看并管理任务结果以及详细信息。GoldenDB 的备份恢复是全局一致性的，备份时控制获取租户的全局一致快照数据存储到本地磁盘或者远端存储设备，恢复时则将数据恢复至一致性时刻点。备份支持多种备份存储媒介，如本地存储、NAS 存储、S3/COS 对象存储等。

➢ 管理备份任务，包括备份任务的创建和配置。选择菜单[租户管理]，在左侧实例树中，单击租户实例，进入实例信息界面，切换到[备份恢复→备份管理]页签

图 35 GoldenDB 备份管理

● 单击**新增**按钮，弹出新增备份任务窗口。备份任务有实时和定时两种，设置定时备份可以在所设定的时间中自动触发备份。以实时备份为例，设置备份任务参数，单击**确定**按钮

图 36 新增备份任务

● 备份管理页面可看到新增的备份任务以及任务状态

图 37 GoldenDB 备份任务状态

- 单击备份进度，进入详情业可查看到进度百分比

图 38 GoldenDB 备份任务进度

- 备份任务执行完后，在备份管理页面单击右侧的任务结果，进入任务结果界面，查看备份结果的详细信息

图 39 备份任务结果

➢ 管理恢复任务，主要包括创建恢复任务、配置恢复参数、执行恢复任务。选择菜单[租户管理]，在左侧实例树中，单击租户实例，进入实例信息界面，切换到[备份恢复→恢复管理]页签

图 40 GoldenDB 恢复管理

- 单击新增按钮，弹出新增恢复任务窗口，进入恢复配置选择，以分片级的恢复为例，如下图

表 15 恢复配置选择参数说明

参数	说明
检测 Binlog 备份时间	开启后将检测 Binlog 文件数据是否满足备份要求。
一致性回滚	是否一致性回滚是指恢复完成后是否回滚掉备份时存在的活跃事务。

图 41 创建恢复任务

- 单击下一步按钮，进入备份文件选择界面，可选择备份开始时间进行备份文件的查询，建议尽量选择结束时间最靠近恢复时间的备份任务产生的文件

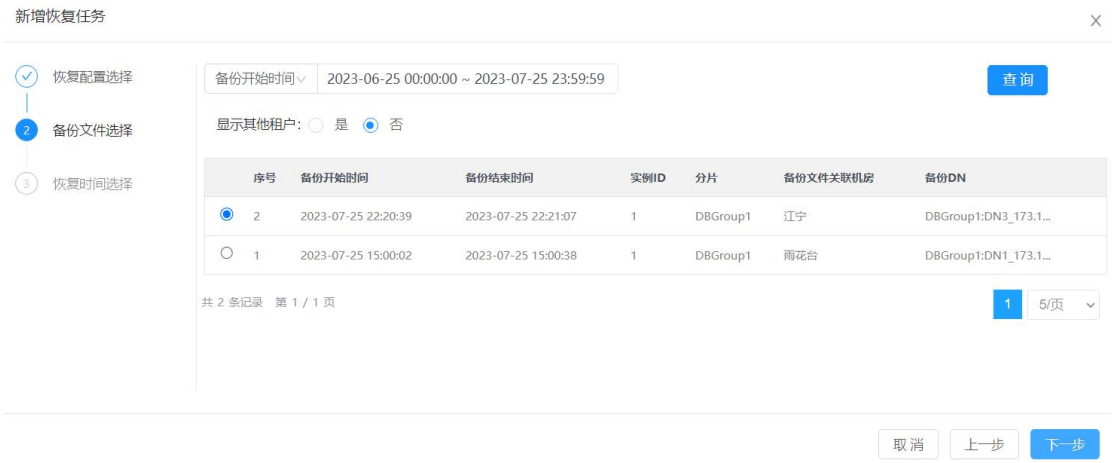


图 42 选择合适的备份数据文件

- 选择待恢复的备份文件，单击下一步按钮，进入恢复时间选择界面。选择好恢复时间后，单击确定按钮，开始恢复任务

图 43 指定恢复时间

- 恢复管理页面可看到新增的恢复任务以及任务状态

图 44 恢复任务状态

- 恢复任务完成后，单击任务结果按钮，进入任务结果界面，查看恢复结果详细信息

图 45 数据恢复结果

5.5. 分布式死锁和锁等待检测

GoldenDB 会对业务访问数据库过程中发生的死锁和锁等待进行检测，通过 insight 运维平台可以方便地查看死锁信息以及不同实例下不同分片的锁等待统计数据。死锁分为单点死锁和分布式死锁，发生在单个 DN 上的死锁称为单点死锁，多个 DN 之间互相等待对方持有的锁称为全局分布式死锁。

➢ DN 死锁检测。选择菜单[统计监控→诊断→死锁检测→DN 死锁]，进入 DN 死锁检测界面

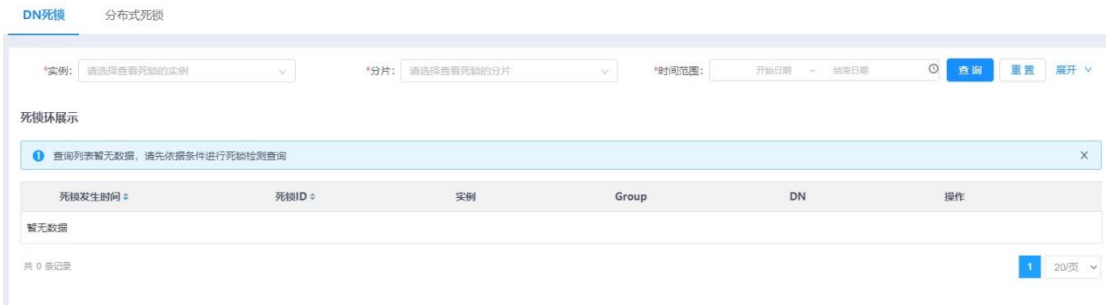


图 46 DN 死锁检测页面

- 单击展开按钮，展开查询条件（包括：实例、分片、时间范围、事务 ID、GTM GTID、SQL 语句、DN 线程 ID、死锁 ID），其中实例、分片、时间范围必填。

图 47 输入合适的查询条件

- 设置查询条件，单击查询按钮，单击列表左侧三角按钮，可展开所选数据的死锁环信息。

图 48 死锁信息查询结果

- 单击死锁环上的详情按钮，展开死锁的详细信息

图 49 死锁相关的详细信息

➢ 分布式死锁检测。选择菜单[统计监控→诊断→死锁检测→分布式死锁]，进入分布式死锁检测界面

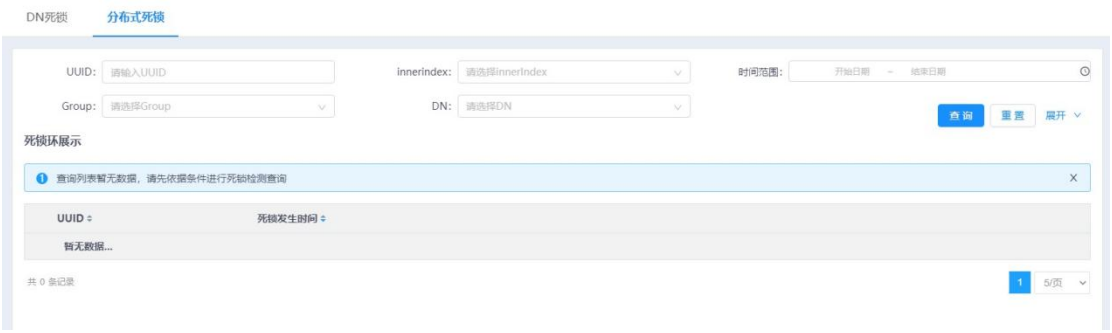


图 50 分布式死锁检测页面

- 单击展开按钮，展开查询条件（包括：UUID、innerindex（死锁环顺序）、时间范围、Group、DN、阻塞表、等锁方 gtid、持锁方 gtid、等锁语句、持锁事务 ID）

图 51 输入合适的查询条件

- 设置查询条件，单击查询按钮，单击列表左侧三角按钮，可展开该分布式死锁的详细信息

图 52 死锁查询结果

➢ 锁等待检测。选择菜单[统计监控→诊断→锁等待]，进入锁等待检测

界面

图 53 指定检测锁等待的对象

- 选择右上角的时间范围，查询到不同时间范围内的数据。可手动自定义选择一段时间，也可快速选择近 1 小时/天/周，默认选择近 1 小时。
- 选择实例、相应分片，单击查询按钮，可查询对应锁等待统计数据。

图 54 锁等待趋势图

- 单击锁等待数量排行，可查看 DN 的锁等待数量变化趋势图，鼠标悬停在图上或者单击数据点，可查看相应时刻的详细数据信息

5.6. 慢日志分析

通过 insight 运维平台可以方便地对系统运行过程中的慢日志进行分析，包括查看慢 SQL 的变化趋势、数量明细以及分析慢 SQL 在执行过程

的各个阶段耗时详情。

➢ 查看慢 SQL 数量变化趋势。选择菜单[统计监控→诊断→慢日志分析]，进入慢日志分析界面

图 55 GoldenDB 慢日志分析页面

● 在慢日志分析界面，选择右上角的时间范围，查询指定时间范围内的数据。可手动自定义选择一段时间，也可快速选择近 1 小时/天/周，默认选择近 1 小时

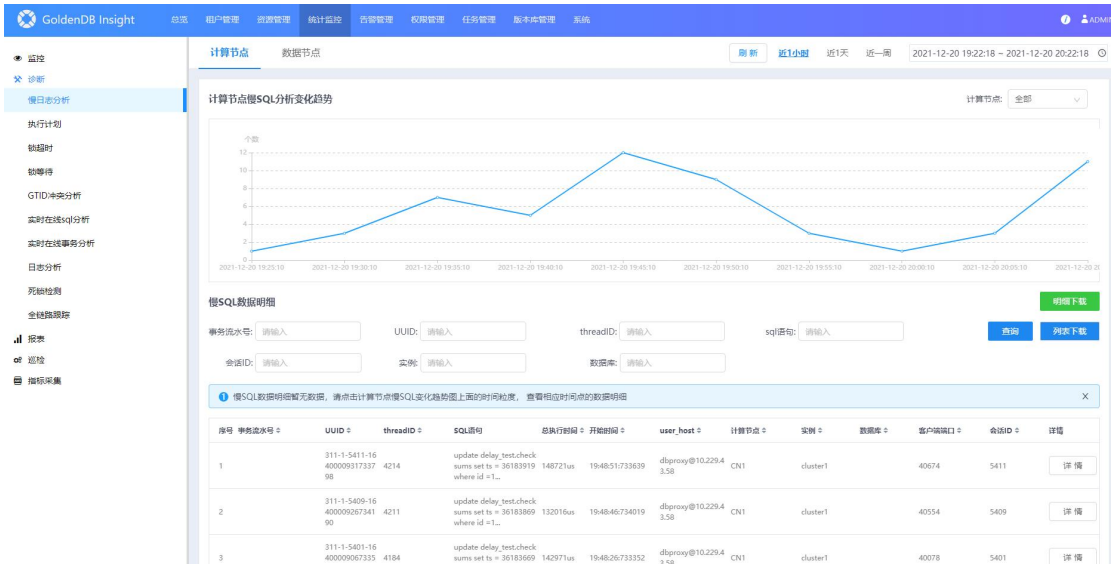


图 56 慢 SQL 趋势图

- 对慢日志分析界面上半页的慢 SQL 分析变化趋势图，以计算节点为例，可进行下列操作

表 16 慢日志操作说明

如果	那么
查询相应 CN 慢 SQL 分析变化趋势	在右上角**计算节点**中选择对应计算节点。
查看相应时刻的详细数据信息	鼠标悬停在图上，单击数据点可查看该时刻慢 SQL 数据。
下载明细	单击**明细下载**按钮。

➢ 分析慢 SQL 各阶段耗时。选择菜单[统计监控→诊断→慢日志分析]，进入慢日志分析界面，在下半页可以看到慢 SQL 数据明细

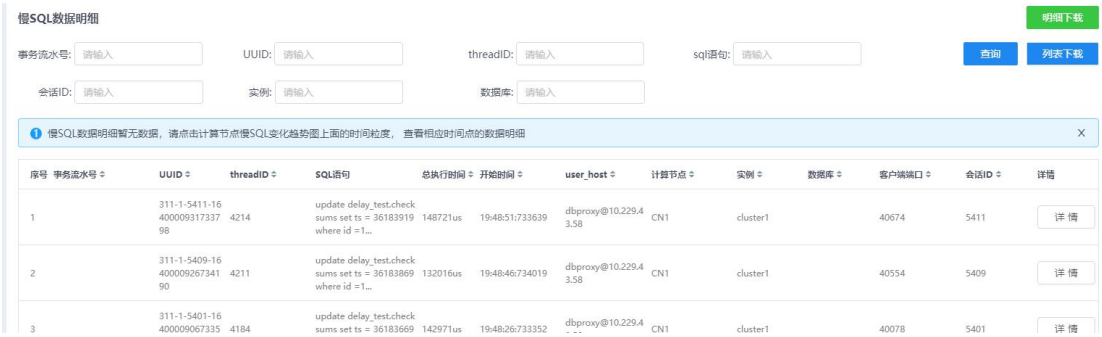


图 57 慢 SQL 明细查询

- 在待查看的慢 SQL 行，单击右侧详情按钮，进入详情界面，查看所选慢 SQL 语句的执行时间拆解

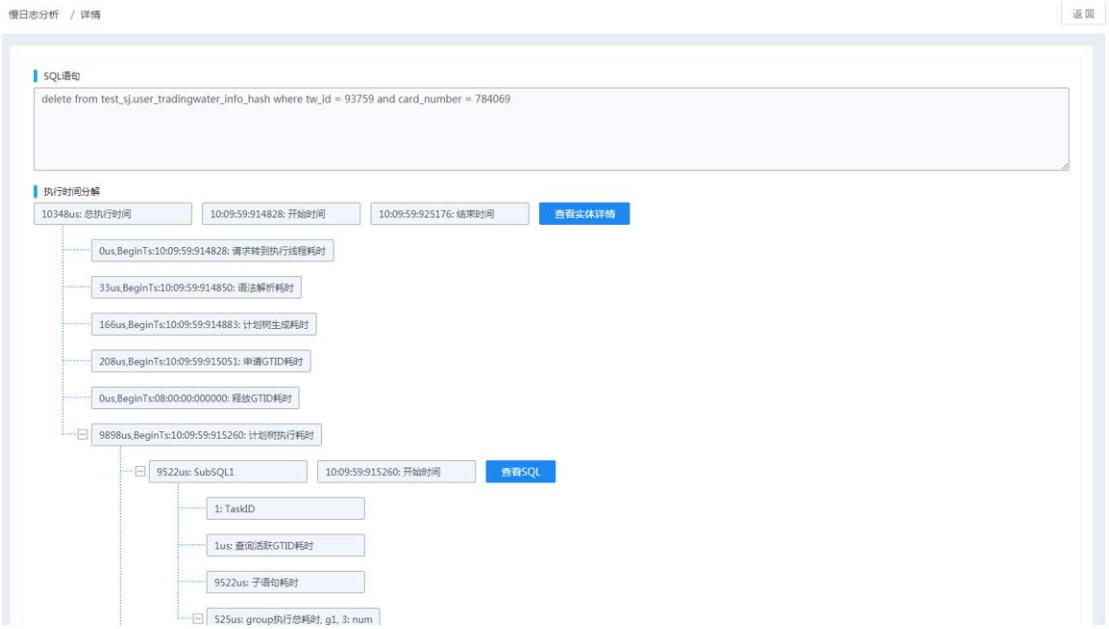


图 58 SQL 执行开销统计树

- 单击执行时间分解下的查看实体详情按钮，查看该执行阶段对应组件的详情信息，单击查看 SQL 按钮，查看该执行阶段拆分出来的 SQL 语句的详情信息

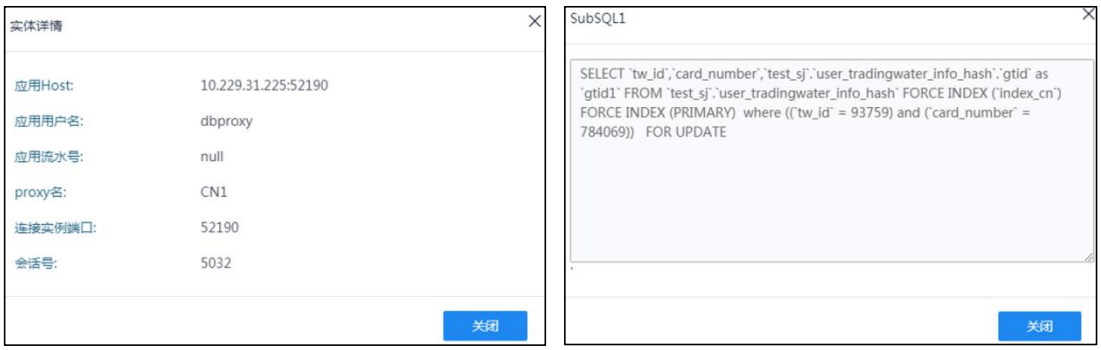


图 59 查看分阶段执行的详情信息

5.7. 性能问题排查

insight 运维平台提供了多种手段辅助用户进行数据库性能问题排查，

包括在线 sql 分析和查杀、在线事务分析和查杀、分布式事务 GTID 冲突分析、全链路跟踪、热点监控、TOP SQL 统计。

➢ 在线 sql 分析和查杀。使用实时在线 sql 分析功能，用户可查询指定计算节点的会话信息，并且对发现的异常 sql 提供了查杀功能。选择菜单[统计监控→诊断→实时在线 sql 分析]，进入实时在线 sql 分析界面

图 60 实时在线 SQL 分析

● 选择实例和相应计算节点，输入语句执行时长阈值，单击查询按钮。该功能可帮助用户定位到执行时间超过该阈值未结束的语句

图 61 通过合适条件过滤在线 SQL

● 单击在线 sql 列表表头各列的排序按钮，可对各项数据进行排序

- 对于筛选出来的异常 sql，用户可进行查杀。单击右侧操作列的 KILL 按钮，在弹出的确认框中单击确定按钮，可完成对异常语句所在的事务的查杀。除了 KILL 外，运维平台还提供了将异常语句拉入黑名单的功能，单击加入黑名单按钮，在弹出的确认框中单击确定按钮
- 上述 KILL 和拉入黑名单功能支持对多条异常 sql 批量操作。勾选指定行前面的方框，点击批量 KILL 事务按钮或者批量加入黑名单按钮，即可进行批量操作

图 62 对 SQL 进行干预

- 单击 KILL 事务历史记录按钮，进入历史记录页面，可查看被查杀掉的异常 sql 历史信息

图 63 干预 SQL 的操作记录在历史列表中

➢ 在线事务分析和查杀。使用实时在线事务分析功能，用户可查询指定计算节点的会话信息，并且对发现的异常事务提供了查杀功能。选择菜单[统计监控→诊断→实时在线事务分析]，进入实时在线事务分析界面

图 64 通过合适条件过滤实时在线事务

- 选择实例和相应计算节点，输入事务持续时长的阈值，单击查询按钮。该功能可帮助用户定位到执行时间超过该阈值未结束的事务
- 单击在线事务列表表头各列的排序按钮，可对各项数据进行排序
- 对于筛选出来的异常事务，用户可进行查杀。与异常 sql 查杀操作方法相同，并且支持对多个异常事务进行批量操作
- 单击 KILL 事务历史记录按钮，进入历史记录页面，可查看被查杀掉的异常事务历史信息

图 65 干预事务的操作记录在历史列表中

➢ 分布式事务 GTID 冲突分析。当发生分布式 GTID 冲突时，业务性能将下降，insight 运维平台提供了分布式 GTID 冲突分析功能，帮助用户在分析性能问题时能方便地掌握分布式事务冲突的比例与趋势，以及冲突事务与被冲突事务详细信息。选择菜单[统计监控→诊断→GTID 冲突分析]，进入 GTID 冲突分析界面

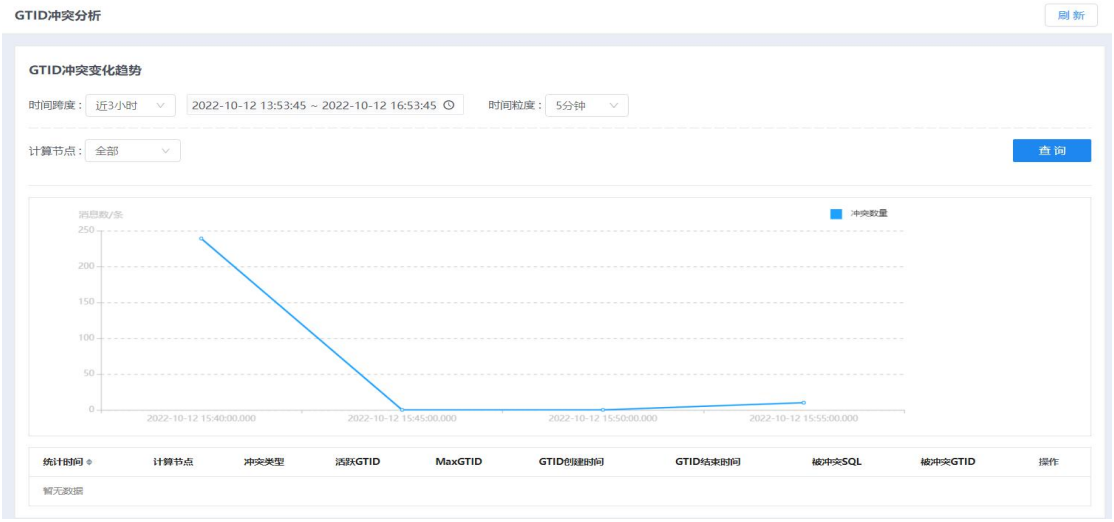


图 66 GTID 冲突分析页面

● 选择时间跨度、时间粒度以及计算节点，单击搜索按钮，查询指定时间范围内的 GTID 冲突变化趋势。可手动自定义选择一段时间，也可快速选择近 1 小时/天/周，默认选择近 1 小时

● 鼠标悬停在图上的数据点，可查看该点的详细数据信息。单击图上某一数据节点，下方会出现该点的详细数据列表。可根据统计时间对数据进行排序

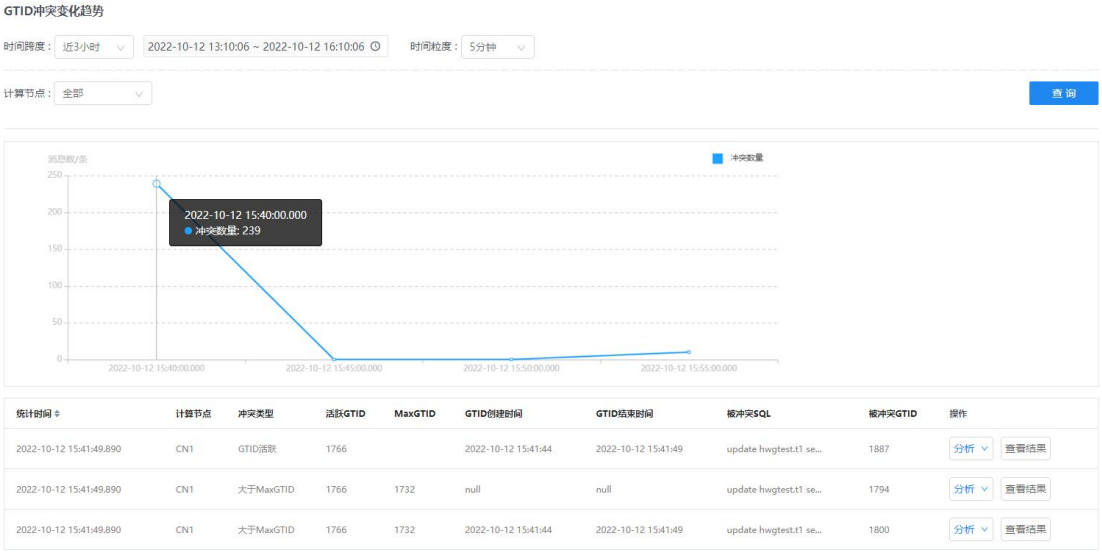


图 67 GTID 冲突变化趋势图

- 鼠标悬停在被冲突 SQL 上，可以查看发生分布式 GTID 冲突时具体的被冲突 SQL

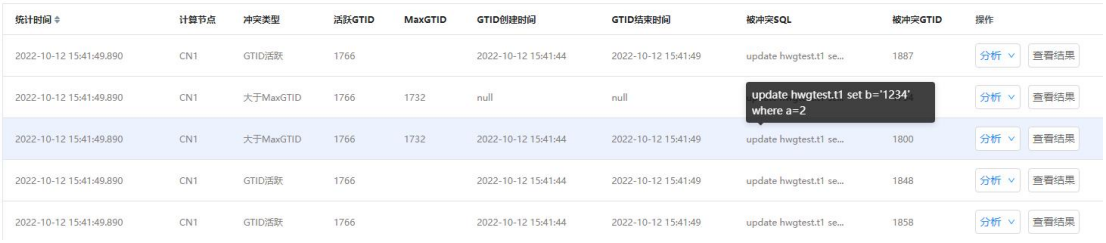


图 68 查看 GTID 冲突的关联 SQL

- 单击右侧分析按钮，在分析完毕后单击查看结果按钮可以查看具体阻塞 SQL 与被冲突 SQL 的信息

图 69 查看阻塞/被阻塞 SQL 的冲突信息

➢ 全链路跟踪。insight 运维平台支持业务语句在 GoldenDB 内部执行流程的全链路追踪包括“CN 收到语句—CN 预处理—CN 路由给 DN—DN 处理—CN 二次处理—CN 返回结果”完整的链路中各个环节的执行情况。选择菜单[统计监控→诊断→全链路跟踪]，进入全链路跟踪界面

图 70 GoldenDB 全链路跟踪页面

● 单击展开按钮，展开筛选条件（包括：原始 SQL、事务开始时间、事务结束时间）。设置好筛选条件后，点击查询按钮。该功能可帮助用户排

查业务响应性能异常时数据库侧的全链路运行情况，用户可以将业务响应性能异常的时间段作为查询条件，也可将其看到的响应时延大的 SQL 作为查询条件

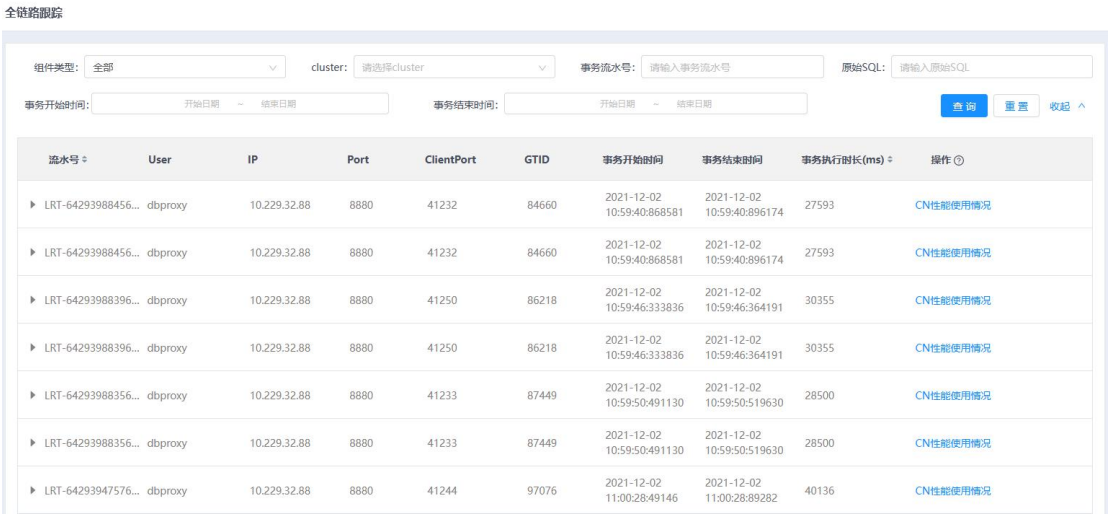
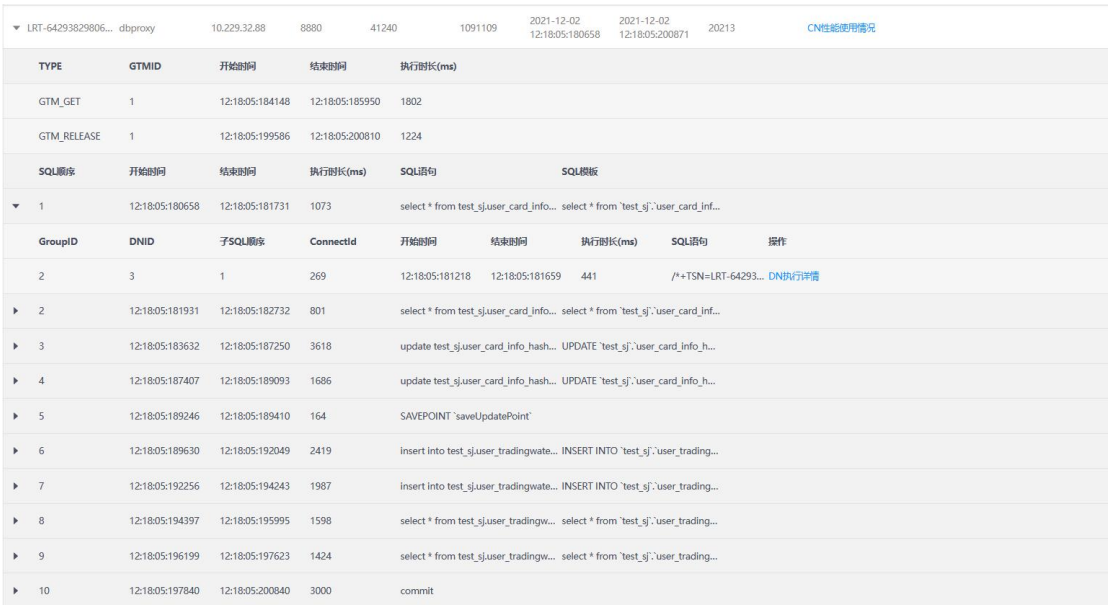


图 71 全链路跟踪列表

- 单击结果行右侧的 CN 性能使用情况可跳转至[租户管理→节点→性能]页面
- 单击查询结果行左侧的三角按钮，展开查询结果详情。可以看到该 sql 执行的全链路跟踪分析结果（SQL 顺序、执行时间、SQL 语句等相关信息），再单击展开的结果详情中的三角按钮，可以看到更加详细的 GroupID、DNID、子 SQL 顺序、执行时长等内容



LRT-64293829806...	dbproxy	10.229.32.88	8880	41240	1091109	2021-12-02 12:18:05:180658	2021-12-02 12:18:05:200871	20213	CN性能使用情况

TYPE	GTMID	开始时间	结束时间	执行时长(ms)
GTM_GET	1	12:18:05:184148	12:18:05:185950	1802
GTM_RELEASE	1	12:18:05:199586	12:18:05:200810	1224

SQL顺序	开始时间	结束时间	执行时长(ms)	SQL语句	SQL模板
1	12:18:05:180658	12:18:05:181731	1073	select * from test_sj.user_card_info...	select * from `test_sj`.`user_card_inf...

GroupID	DNID	子SQL顺序	ConnectId	开始时间	结束时间	执行时长(ms)	SQL语句	操作
2	3	1	269	12:18:05:181218	12:18:05:181659	441	/*+TSN=LRT-64293...	DN执行详情

2	12:18:05:181931	12:18:05:182732	801	select * from test_sj.user_card_info...	select * from `test_sj`.`user_card_inf...
3	12:18:05:183632	12:18:05:187250	3618	update test_sj.user_card_info_hash...	UPDATE `test_sj`.`user_card_info_h...
4	12:18:05:187407	12:18:05:189093	1686	update test_sj.user_card_info_hash...	UPDATE `test_sj`.`user_card_info_h...
5	12:18:05:189246	12:18:05:189410	164	SAVEPOINT `saveUpdatePoint`	
6	12:18:05:189630	12:18:05:192049	2419	insert into test_sj.user_tradingwate...	INSERT INTO `test_sj`.`user_trading...
7	12:18:05:192256	12:18:05:194243	1987	insert into test_sj.user_tradingwate...	INSERT INTO `test_sj`.`user_trading...
8	12:18:05:194397	12:18:05:195995	1598	select * from test_sj.user_tradingw...	select * from `test_sj`.`user_trading...
9	12:18:05:196199	12:18:05:197623	1424	select * from test_sj.user_tradingw...	select * from `test_sj`.`user_trading...
10	12:18:05:197840	12:18:05:200840	3000	commit	

图 72 全链路跟踪详情列表

- 单击上图中的 DN 执行详情，展示该子 SQL 语句在 DN 上的执行情况，包括 DBID、执行时长、SQL 语句等信息

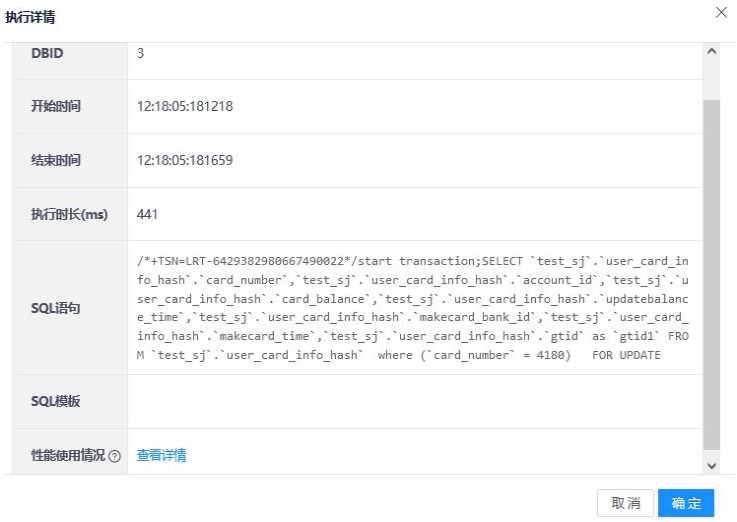


图 73 在分布式组件内执行步骤详情

- 热点监控。insight 运维平台会对数据库中被频繁访问的热点数据进行监控和统计，并在运维平台上展示热点表读写操作数量，可帮助用户观

察和分析热点表的负载情况，为用户处理热点性能问题提供基础监控数据，便于用户快速调整对应的热点业务。选择菜单[统计监控→监控→热点更新数据统计]，进入热点更新数据统计界面

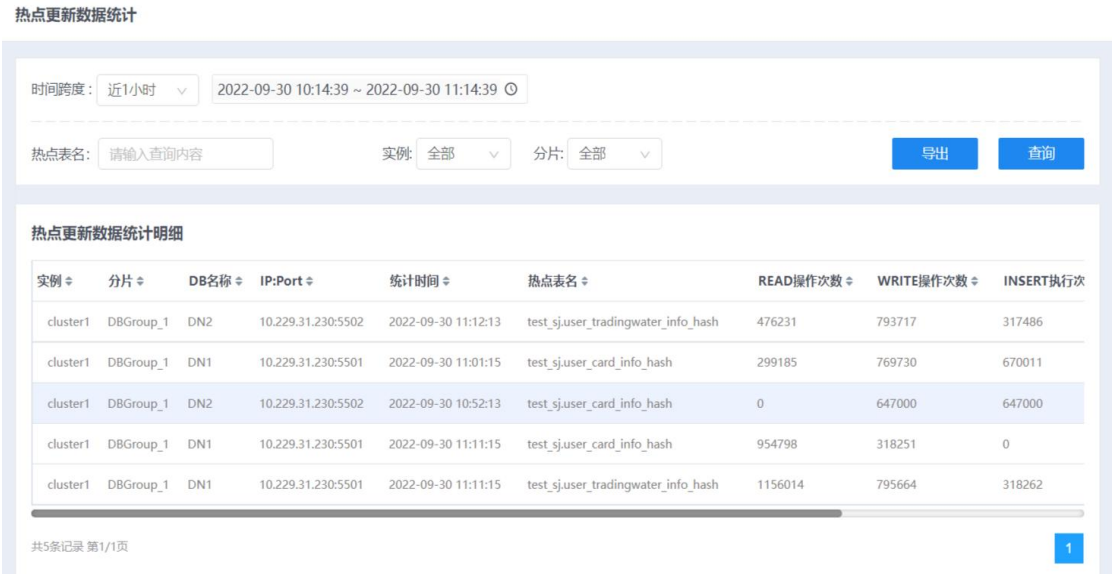


图 74 热点更新数据统计

- 选择时间跨度、实例等信息，单击查询按钮，查询符合条件的统计数据。从统计数据中可以获取热点表的读写次数以及具体的 insert、update、delete 次数。左上角的时间范围，手动自定义选择一段时间，也可快速选择近 1 小时/天/周，默认选择近 1 小时
- 可根据热点表名信息更精确的定位热点读写操作次数。在热点表名搜索框中输入想要查询的热点表名，点击查询按钮

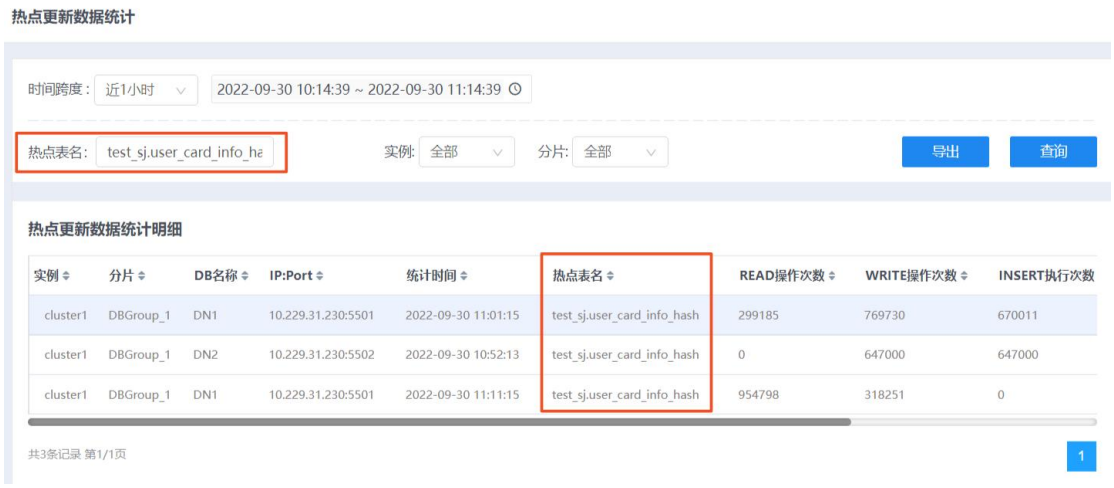


图 75 统计具体表的热度

● 在热点更新数据统计明细中，单击表头各列的排序按钮，可对各项数据进行排序，以统计时间列为例，单击右侧的三角按钮，可对热点数据按照统计时间进行升序/降序排序

图 76 热点数据多维度排序

➢ TOP SQL 统计。TOP SQL 是性能问题诊断和排查过程中的一个重要指标，insight 运维平台会对业务语句的执行情况进行统计，按照不同指标进行 TOP 排行，包括执行总时间最长，平均执行时间最长，执行次数最多这三个维度。选择菜单[统计监控→监控→TOPN]，进入 TOPN 界面

图 77 TOP SQL 统计

● 选择右上角的时间范围，可查询到指定时间范围内的 TOP SQL 数据。可手动自定义选择一段时间，也可快速选择近 1 小时/天/周，默认选择近 1 小时

● 选择不同指标（包括总执行时间、平均执行时间、执行次数），以及 Top 数（可选 5、10、30），可进行 TOP SQL 数据筛选展示。

5.8. 版本升级

通过 insight 运维平台可对 GoldenDB 组件版本进行升级。

➢ 配置版本包。设置版本库各项参数，并上传版本到指定服务器目录下。如下图：

图 78 GoldenDB 在线升级版本

➢ 执行版本升级。选择菜单，进入版本升级管理界面，可以执行版本升级任务，如下图：

图 79 在线版本升级任务状态

6. 提升数据库安全

随着金融机构对数据库应用系统依赖程度的日益增强，数据库安全受到普遍关注，需重视和实施数据库系统安全保护措施，包括网络安全、账号安全、数据安全、存储和传输安全，通过数据库审计加以防范和保护。

6.1. 网络规划

按业务重要程度对网络区域进行隔离划分，设置网络接入区、业务服务区、内部系统区和密码服务区。

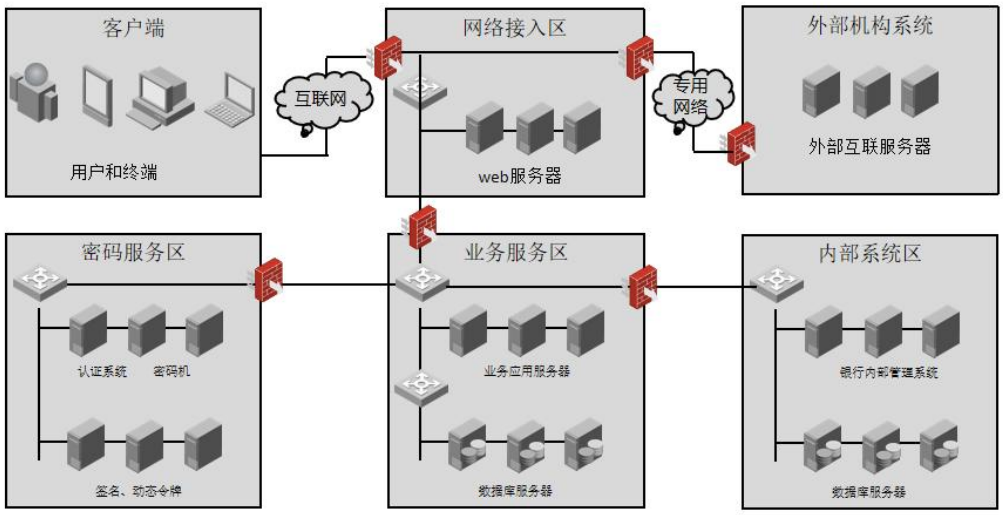


图 80 网络区域划分

用户终端和营业网点通过互联网接入网络接入区，在网络接入区设置防火墙设备，外部机构系统也通过防火墙设备接入网络接入区。

网络接入区部署 Web 服务器，对外提供外部服务请求，并将请求送到业务服务区处理。

内部重要的业务服务系统划分独立的区域，前端的 web 服务器通过防火墙接入业务服务器。银行内部管理系统分开部署，需要访问业务服务区时，通过防火墙访问业务应用系统。

业务应用系统和数据库服务器部署在同一个区域，直接通过内部交换机互联。

关键的密码服务、认证服务独立部署在内部区域，通过防火墙进行网络隔离，提升核心密码服务的安全性。

6.1.1. GoldenDB 端口列表

数据库业务服务区内的每台服务器，设置开放端口，仅打开需要互联访问的端口，未使用到的服务端口，在服务器上关闭，提高内部各服务器的安全性。

GoldenDB 数据库服务器需要开放的端口列表如下：

协议类型	服务端口	服务节点	访问节点
TCP/IP	8880	计算节点	应用服务器
TCP/IP	3306	数据节点	管理节点， 数据节点， 计算节点
TCP/IP	5018， 5600， 6151-6200，	数据节点	数据节点， 管理节点
TCP/IP	21/22， 3306， 5012， 5518， 8021，	数据节点	数据节点

	16313, 6251-6300,		
TCP/IP	21/22, 5010, 5600, 8021, 16313 6026-6030, 8501-8600,	GTM	管理节点, GTM, 计算节点
HTTP	8444	管理节点	浏览器访问 Insight 页面
TCP/IP	3309	管理节点（RDB）	管理节点
TCP/IP	5004，6406-6410	管理节点（MDS）	管理节点, GTM, 数据节点, 计算节点
TCP/IP	5006，6006-6010	管理节点（PM）	管理节点, 计算节点
TCP/IP	5008，6016-6020	管理节点（CM）	管理节点, 数据节点

表 17 GoldenDB 开放端口列表

6.2. 主机账号安全设计

定期回收/重置主机账号的密码，例如 root 密码，可以通过专用的主

机账号管理系统进行回收和重置。如果需要使用 root 密码（GoldenDB 安装时需要使用到 root 密码），通过专用申请流程进行申请，安装完成后及时收回。

GoldenDB 数据库组件在主机上创建的用户密码，按安装部署后统一设置，或有专用账号管理系统定期设置。

6.3. 数据库账号安全设计

访问数据库账号进行权限划分，仅给各账号必要的权限范围,一般设置系统管理员、应用账号，可设置独立的审计管理员账号。GoldenDB 支持数据库账号、表对象的授权，通过分权进行安全控制。

- 系统管理员账号：只创建表、将表权限赋予应用账号，不具备对表数据的增删改查权限。
- 应用账号：只有数据库表的访问权限，对数据库表进行增删改查，不具备建表权限。

通过 GRANT/REVOKE 命令对应用用户授权和回收，如：

GRANT UPDATE, DELETE, INSERT, SELECT on app_xxx_1 TO app_user1;

REVOKE UPDATE, DELETE, INSERT, SELECT on app_xxx_2 FROM app_user1。

6.4. 数据库传输加密

通过传输层加密，避免应用服务器访问数据库时使用明文协议传输，

减少网络攻击和窃取数据风险。要启用链路层加密,进行服务器证书生成,命令行客户端和 jdbc 进行关联配置。

6.4.1. 使用 mysql 客户端 ssl 连接 GoldenDB

```
mysql --ssl-ca="/home/cert/ca.pem" --ssl-cert="/home/cert/client-cert.pem" --ssl-key="/home/cert/client-key.pem" -h[GoldenDB 服务器 IP] -uroot -p123456 -P1111
```

其中标为红色的项需要替换成现场使用的具体路径，包括 ca.pem 的文件位置，client-cert.pem 的文件位置，client-key.pem 的文件位置，ip，用户名，密码，端口号。

6.4.2. 使用 jdbc ssl 连接 GoldenDB

1. 生成 truststore 文件

keytool -import -alias mysqlServerCACert -file ca.pem -keystore truststore

输入密码

输入 y 生成 truststore 文件

把 truststore 文件复制到合适的地方

2. java 中的连接代码

```
static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
static final String DB_URL = static final String DB_URL =
"jdbc:mysql://[GoldenDB 服务器 IP]:[端口号]/[数据库
```

名]?verifyServerCertificate=true&useSSL=true&requireSSL=true";

在 JAVA 代码中添加以下两条代码

...

System.setProperty("javax.net.ssl.trustStore","[truststore 的文件位置]");

System.setProperty("javax.net.ssl.trustStorePassword","[生成 truststore 文件时输入的密码]");

...

6.5. 加密存储

GoldenDB 支持表级别的透明加密，对敏感的表数据，对整表进行加密存储。在建表时设置 encryption='Y'属性，如下：

```
MySQL [d_6_3_3_un]> create table t1_en (a int key)encryption='Y' distributed by hash(a)(g1,g2,g3) ;
Query OK, 0 rows affected (0.01 sec)
```

建表时支持选择国密加密，如下：

```
MySQL [d_6_3_3]>
MySQL [d_6_3_3]> create table t_6_3_3_2 (id int key,name varchar(10)) encryption='sm4';
Query OK, 0 rows affected (0.03 sec)
```

GoldenDB 支持字段级的函数加密，如应用的用户密码，可以通过 AES_ENCRYPT()函数进行加密后存储，SQL 语句示例如下：

```
MySQL [d_6_3_3]>
MySQL [d_6_3_3]> insert into t_6_3_3_1 values(1,AES_ENCRYPT('guessme', 'salt'));
Query OK, 1 row affected (0.00 sec)
Records: 1  Duplicates: 0  Warnings: 0

MySQL [d_6_3_3]>
```

GoldenDB 数据库用户密码采用国密加密方式。

6.6. 加强数据库审计

数据库内部管理和访问，包括数据库 DBA、操作人员、开发人员等，

容易出现权限滥用、高危操作、误操作、低质量 SQL，引起数据库故障和可用性问题，影响数据库平稳运营；同时面临外部访问威胁，包括恶意访问、账号密码截取，甚至黑客攻击，如 SQL 注入、零日攻击等，危害数据库数据安全。加强数据库审计，对数据库访问进行监控和告警，可以有效控制或记录数据库访问风险，进行数据库安全评估和迭代加固，保障日常安全运行。

GoldenDB 数据库审计对数据库操作进行细粒度审计，实时记录数据库的活动，对数据库遭受到的风险行为进行记录、告警，方便进行回溯和改进。如：数据库漏洞攻击、SQL 注入攻击、高危风险操作等;以及系统管理员角色或用户登录，执行用户创建、用户权限变更、用户删除等操作；普通用户执行的数据表、存储、用户的数据对象变更等操作；执行 DDL、DML、DCL、参数修改等重要操作。

审计日志需要开启 GoldenDB 审计开关控制项，可以设置审计的类型，如下是审计主要配置项：

```
#审计日志功能开关
#unit: NA, range: 0-OFF,1-ON, default: 0, dynamic: yes;
server_audit_logging = 1

#审计日志参数配置：配置为空记录所有事件，默认记录STARTUP,CONFIGURE,CONNECT;
#支持以下事件，多个事件配置时按逗号分隔事件
#   STARTUP                 ; CONFIGURE
#   CONNECT                 ; QUERY
#   QUERY_DDL               ; QUERY_DML
#   QUERY_DML_NO_SELECT     ; QUERY_DCL
#unit: NA, range: NA, default: STARTUP,CONFIGURE,CONNECT, dynamic: yes;
server_audit_events = STARTUP,CONFIGURE,CONNECT,QUERY
```

图 81 GoldenDB 审计配置项

开启审计日志后，GoldenDB 在运行过程中记录数据库操作记录到审计日志中，如下是一段数据库审计日志的样例：

```
10-19 19:34:23:790 F[dbp_proxytool.cc] L[3533] 2022-10-19 19:34:23,CN2_10.69.41.91_6451,dbproxy,10.69.41.90:62772,8880_1476,,QUERY,1,commit,0
10-19 19:34:23:790 F[mdc_conn.cc] L[2879] 2022-10-19 19:34:23,DBPROXY_10.69.41.91_6451,dbproxy,10.69.41.90,8880_1476,,DISCONNECT,1,,0
10-19 19:34:26:380 F[mdc_conn.cc] L[2879] 2022-10-19 19:34:26,DBPROXY_10.69.41.91_6451,dbproxy,10.69.41.90,8880_1477,,CONNECT,1,,0
10-19 19:34:26:380 F[mdc_conn.cc] L[2879] 2022-10-19 19:34:26,DBPROXY_10.69.41.91_6451,dbproxy,10.69.41.90,8880_1477,,DISCONNECT,1,,0
10-19 19:34:26:700 F[dbp_proxytool.cc] L[3533] 2022-10-19 19:34:26,CN2_10.69.41.91_6451,dbproxy,10.69.41.90:62468,8880_1474,,QUERY,1,create user zte identifie
d by'******',0
10-19 19:34:26:850 F[mdc_conn.cc] L[2879] 2022-10-19 19:34:26,DBPROXY_10.69.41.91_6451,dbproxy,10.69.41.90,8880_1478,,CONNECT,1,,0
10-19 19:34:26:910 F[dbp_proxytool.cc] L[3533] 2022-10-19 19:34:26,CN2_10.69.41.91_6451,dbproxy,10.69.41.90:62468,8880_1474,,QUERY,1,grant all ON poc_test.* T
O 'zte',0
10-19 19:34:27:100 F[dbp_proxytool.cc] L[3533] 2022-10-19 19:34:27,CN2_10.69.41.91_6451,dbproxy,10.69.41.90:63056,8880_1478,,QUERY,1,grant SELECT on performan
ce_schema.metadata_locks to 'zte'@'%' gdb_nolock,0
10-19 19:34:27:100 F[mdc_conn.cc] L[2879] 2022-10-19 19:34:27,DBPROXY_10.69.41.91_6451,dbproxy,10.69.41.90,8880_1478,,DISCONNECT,1,,0
10-19 19:34:27:130 F[mdc_conn.cc] L[2879] 2022-10-19 19:34:27,DBPROXY_10.69.41.91_6451,dbproxy,10.69.41.90,8880_1479,,CONNECT,1,,0
10-19 19:34:27:130 F[dbp_proxytool.cc] L[3533] 2022-10-19 19:34:27,CN2_10.69.41.91_6451,dbproxy,10.69.41.90:63180,8880_1479,,QUERY,1,grant cn_session_variable
s_admin on *.* to 'zte'@'%' gdb_nolock;,1064
```

图 82 GoldenDB 审计日志

审计用户也可以登录 GoldenDB Insight 管理台开启数据库审计，通过 Insight 页面可以审计数据库相关操作。包括：查看审计日志，核对上述事件的时间、主体标识、客体标识、结果等信息。

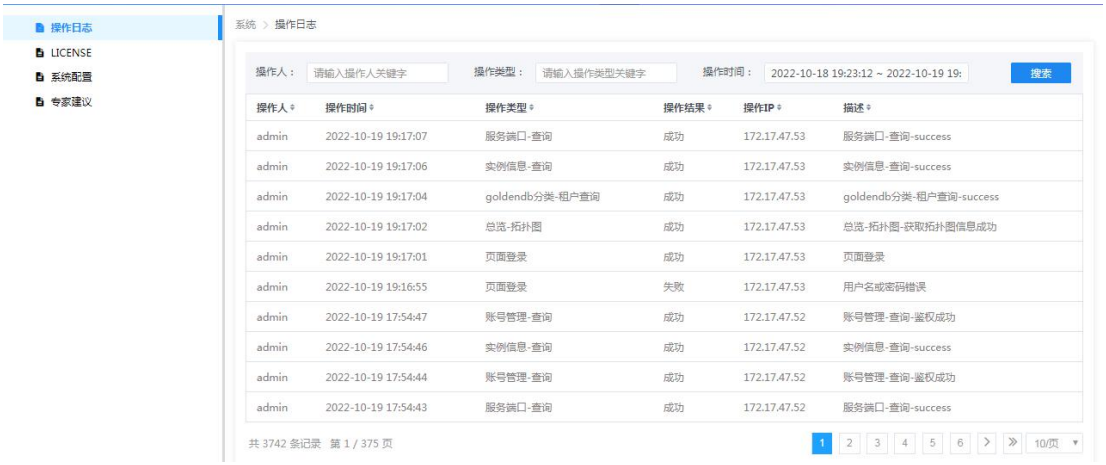


图 83 GoldenDB Insight 查阅审计日志内容

6.7. 数据脱敏

应用数据库上存储个人姓名、身份证号、住址等个人敏感信息，以及企业金融数据、个人财务数据等。从应用数据库进行数据迁移、数据导出进行数据分析、问题定位、测试环境搭建等用途时，需要通过有效手段对敏感数据进行脱敏处理，屏蔽或改写敏感字段，避免信息泄露。例如用*

代替姓名或身份证特定位数，改写资产金额等信息。

GoldenDB 支持在数据导出时设置脱敏策略，包括：证件保留尾号脱敏、邮箱地址前缀脱敏、邮箱地址前缀+域名脱敏、数字脱敏、乱序脱敏、随机字符替代脱敏、正则表达式脱敏、全遮蔽脱敏，下面是设置某表脱敏策略的语法：

```
MySQL [maskrule_test]> create maskrule on table t1 test rule[ concat(left(id,2),'***',right
(id,2))as id,concat(left(idcard,3),'******',right(idcard,4))as idcard,'###' as sname,concat
('****',right(email,10))as email ];
Query OK, 0 rows affected (0.024 sec)
```

图 84 数据库脱敏策略设置

设置上述脱敏规则后，再执行该表查询时，结果集进行相应的转换，如下图：

```
MySQL [maskrule_test]> Ctrl-C -- exit!
Aborted
[root@gdb91 ~]# mysql -utest -pdb1x@NJ+1 -h10.69.41.90 -P8880
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MySQL connection id is 109509
Server version: 8.0.18-log Source distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MySQL [(none)]> select * from maskrule_test.t1;
+---------+---------------+-------+---------------------+------+----------------+
| id      | idcard        | sname | date                | year | email          |
+---------+---------------+-------+---------------------+------+----------------+
| 13***67 | 311******1819 | ###   | 2017-09-12 11:13:41 | 2017 | ****@baidu.com |
| 83***15 | 321******7710 | ###   | 2012-04-12 15:46:20 | 2012 | ****st@163.com |
| 23***57 | 481******2220 | ###   | 2014-06-12 11:13:41 | 2014 | ****ldendb.com |
| 78***42 | 320******5213 | ###   | 2010-02-12 11:13:41 | 2010 | ****976@qq.com |
| 88***34 | 311******1121 | ###   | 2016-08-12 11:13:41 | 2016 | ****22@163.com |
| 34***25 | 315******1121 | ###   | 2018-12-12 11:13:41 | 2018 | ****2@sina.com |
| 25***15 | 430******5560 | ###   | 2011-03-12 10:18:45 | 2011 | ****237@qq.com |
| 15***85 | 421******1790 | ###   | 2013-05-12 14:13:53 | 2013 | ****en@126.com |
| 68***24 | 441******1467 | ###   | 2015-07-12 11:13:41 | 2015 | ****ng@163.com |
+---------+---------------+-------+---------------------+------+----------------+
9 rows in set (0.004 sec)

MySQL [(none)]>
```

图 85 数据库脱敏查询示例

6.8. 通过 SQL 审核提升服务质量

不合理、低效 SQL 容易导致应用响应延迟，拖累数据库处理性能，引起数据库整体服务质量下降，需要通过多种策略措施提升 SQL 质量，实现

高效处理。建议采用以下措施：

SQL 审核配套文档和培训。基于 GoldenDB SQL 开发手册、分布式数据库开发规范进行宣贯和培训，指导应用开发人员遵从 SQL 规范进行高效开发。

数据库管理员参考分布式数据库开发规范对应用系统 SQL 进行人工审核。包括：

事前审核。GoldenDB 提供 SQL 审核工具，管理员将 DDL 脚本、应用系统和数据库系统上 SQL 日志导入到 SQL 审核工具中，对数据库对象和 SQL 语句进行静态审核。该审核主要基于规则分析，输出待整改 SQL，如危害性 DDL、过复杂 SQL 嵌套、全表扫描、无索引过滤等等。另外，该工具同时提供对表的分片规则和异构数据库字段类型、SQL 语句进行自动改写功能，支撑数据库自动化迁移。

事后审核。GoldenDB 平台实时抓取和记录 SQL 执行语句、统计 SQL 开销，包括计算节点、数据节点各阶段执行的子过程和执行开销，准实时同步到 GoldenDB Insight 智能分析平台进行 SQL 审核分析，以图形化界面方式展示 SQ 审核结果。包括：

A）危害性 DDL 语句。截获 DROP/ALTER/TRUNCATE/DELETE 语句，记录危害性 DDL 详细信息，包括执行时间、源端地址、数据库用户、原始语句、表结构信息、影响记录数规模等内容。系统管理员通过管理界面查看和下载 DDL 审核内容，提供给业务开发部门进行整改。

B）慢 SQL TOPn。统计高排名慢 SQL 语句、执行时间、执行开销、源端地址和用户等信息，并基于规则给出优化建议，系统管理员通过管理界面查看和下载相关信息给业务开发部门进行优化，系统管理员可以在线调测相关慢 SQL 执行计划，支撑管理员对慢 SQL 给出优化建议。

C）SQL 优化建议。基于优化器分析、执行信息统计，优化规则进行智能化分析，识别待优化 SQL，给出优化建议。例如：识别全表扫描、无索引过滤、失效索引、有索引但回表的查询等等，建议增加表分区(如按天、按小时的时间分区)等等。

D）分布式 SQL 审核。基于表的分片规则和 SQL 执行统计，给出数据分布规则调整的建议。例如：基于 SQL 语句统计出频繁被关联的小表，且变更频度低，判定为字典表，建议该表调整为复制表模式。对存在下压执行的 SQL，但不能完全等效自动优化的关联查询，给出下压优化改写建议等等。

SQL 拦截。支持管理员配置拦截规则，根据拦截规则拒绝执行危害性 SQL 语句，例如 DROP、ALTER、TRUANCATE、DELETE、GRANT、REVOKE 语句，规则粒度对应到用户和表。同时记录 SQL 拦截的详细信息。

7. GoldenDB 性能调优

7.1. 性能分析

旨在提供业务系统遇到性能问题时排查的思路以及涉及 GoldenDB 数

据性能问题时的快速分析和定位操作参考。

1. 确认业务具体表现，如响应时延增加、系统超时还是业务报错返回异常。同时，需要考虑近期是否存在业务场景变更、应用版本/数据库版本变更、是否上线新的联机/批量业务等。
2. 确认资源使用率情况，包括前、中、后端资源使用率，应用服务器是否出现资源（CPU/内存/网络/磁盘 IO）瓶颈、应用中间件是否出现队列积压、数据库服务器（CN/DN/GTM）是否出现资源瓶颈。
3. 检查 GoldenDB Insight 监控告警是否存在性能问题期间告警事件，如线程池积压引起的慢查询会触发线程池积压告警事件。
4. 通过统计分析查看性能问题期间事务数统计、SQL 平均耗时、TOP10 等，初步明确性能问题范围，如是否存在大事务、慢 SQL。
5. 进一步分析慢 SQL，分析计算节点和数据节点慢日志进一步确认出现性能问题的 SQL 语句，通过 EXPLAIN 进一步明确慢 SQL 具体原因，并结合 SQL 优化进一步处理。

   计算节点慢日志示例：

```
[2020-03-18 04:36:03:370]|DEBUG||311-1-4091-
1584477361372642|LRT1133282889123678368|49417|49900|dbp_proxytool.cc:3723||#EXECUTE:Port[7788]Session[4091]TransSerial[LRT1133282889123678368]LinkIP[10.46.182.58]LinkPort[16570]UserName[dbproxy]ProxyName[proxy1]ClusterName[cluster1]TotalExecTime[2003025us]BeginTs[04:36:01:375315]EndTs[04:36:03:378340]SQL[select * from test.user_card_info_hash where card_number=527244 for update]
MsgToExecTime[6us]ParserSQLTime[26us]PlanTreeCreateTime[65us]GetGTIDTime[0us]FreeGtidTime[0us]PlanTreeExecTime[2002865us]
SubSQL[1]:TaskID[1]ExecTime[2002860us]BeginTs[04:36:01:375455]FinishTime[2us]QueryGTID[303us]GroupTime[
g1          num:1, duration:2002497us
  sqltoRoute  num:1,duration:130us
  getTask     num:1,duration:3us
  runSql      num:1,duration:9us
  addEpoll    num:1,duration:2us
  DB          connection_id:47546,duration:2002276us,beginTs:04:36:01:375948
  handleEpoll num:1,duration:0us
  worker      num:1,duration:17us
  restoExec   num:1,duration:48us
]SQL[SELECT
`test`.`user_card_info_hash`.`card_number`,`test`.`user_card_info_hash`.`account_id`,`test`.`user_card_info_hash`.`card_balance`,`test`.`user_card_info_hash`.`updatebalance_time`,`test`.`user_card_info_hash`.`makecard_bank_id`,`test`.`user_card_info_hash`.`makecard_time`,`test`.`user_card_info_hash`.`gtid` as `gtid1` FROM `test`.`user_card_info_hash` where (`card_number` = 527244)  FOR UPDATE]
```

7.2. 参数优化

7.2.1. 主机参数配置

1. CPU。操作系统一般根据客户的需求选择，主流生产系统是红帽（Redhat），信创背景下国产操作系统中使用较多的是麒麟（Kylin）操作系统。系统调优主要围绕CPU、内存、io设备、系统运行参数控制等方面进行，常见的包括：

（1）CPU开启性能模式、高核环境进行CPU绑核；
（2）调整内存清理阈值及分配策略、关闭swap；
（3）调整io及网络相关配置；
（4）调整系统资源限制，例如文件句柄数量限制fs.file-max、以及用户级别的open_files_limit

调整系统参数主要涉及两个配置文件/etc/security/limits.conf和/etc/sysctl.conf有内核参数。核参数的方法：

1.使用echo动态修改，如echo "1" >/proc/sys/net/ipv4/tcp_syn_retries，重启后重置为默认值

2.把参数添加到/etc/sysctl.conf 中，然后执行 sysctl -p 使参数生效，永久生效

CPU 运行于性能模式

```
[root@10 ~]# tuned-adm active
Current active profile: throughput-performance
[root@10 ~]# tuned-adm profile throughput-performance
```

CPU 硬件信息

```
cat /proc/cpuinfo | grep  'physical id' | sort | uniq | wc -l（cpu 个数）
cat /proc/cpuinfo | grep 'cpu cores' | sort | uniq（cpu 核数）
cat /proc/cpuinfo | grep 'processor' | sort | uniq | wc -l（逻辑 cpu）
cat /proc/cpuinfo | grep 'model name' | sort | uniq(cpu 厂商)
```

或者使用 lscpu 命令。

2. 内存

NUMA

numactl --hardware 非一致性内存访问

因为 NUMA 默认的内存分配策略是优先在进程所在 CPU 的本地内存中分配，会导致 CPU 节点之间内存分配不均衡，当某个 CPU 节点的内存不足时，会导致 swap 产生，而不是从远程节点分配内存。这就是所谓的 swap insanity 现象。一般建议在 BIOS 中设置关闭 numa。

irqbalance

service irqbalance status

irqbalance 用于优化中断分配，当启动 irqbalance 服务时，irqbalance 会尽可能地将中断均匀地分发给各个 CPU core.

配置 vm.min_free_kbytes 和 vm.swappiness

[root@10 ~]# echo vm.min_free_kbytes=4194304 >> /etc/sysctl.conf （对于 128G 内存）

[root@10 ~]# echo vm.swappiness=0 >> /etc/sysctl.conf

[root@10 ~]# sysctl -p

关闭 swap

[root@10 ~]# echo 3 > /proc/sys/vm/drop_caches

确认 overcommit_memory（内存分配）策略

[root@10 ~]# cat /proc/sys/vm/overcommit_memory

一般有 0，1，2 三种策略

3. 磁盘

测试 io 性能（iops 和吞吐）。

[root@10 ~]# fio -ioengine=libaio -bs=4k -direct=1 -thread -rw=randwrite -filename=/home/fio_test -name="BS 4k rand write" -iodepth=16 -runtime=60 -size=5G

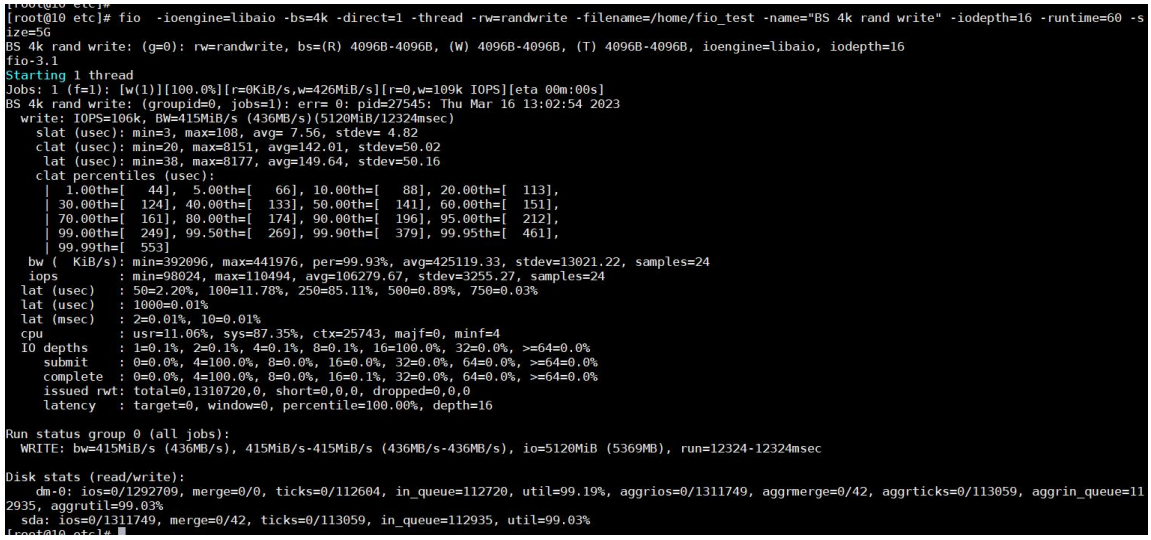


图 86 fio 压测磁盘性能

查看磁盘调度算法

cat /sys/block/$DEVICE-NAME/queue/scheduler

[ ]中即为当前使用的磁盘 IO 调度模式。通常，NVME 盘可以设置为 deadline 或者 none，其他盘设置为 deadline。

```
[gdb1@rh161 ~]$ cat /sys/block/sda/queue/scheduler
noop [deadline] cfq
```

如果需要修改,可以采用 echo 来修改,比如要将 sda 修改为 deadline:

```
# echo deadline > /sys/block/sda/queue/scheduler
```

4. 网络

检查关闭防火墙

```
systemctl status firewalld
systemctl stop firewalld
systemctl disable firewalld
```

检查 ntp 打开，服务器时间已同步

```
systemctl status ntpd
ntpq -p
```

tcp 配置（一键安装时已修改）

GoldenDB 安装时默认修改了（/etc/security/limits.conf 里，hard 类型）打开文件的最大数目 nofile 655360 和进程的最大数目 nproc 65536 和三个系统默认参数（/etc/sysctl.conf 里）

```
net.ipv4.tcp_keepalive_time = 15
net.ipv4.tcp_keepalive_intvl = 2
net.ipv4.tcp_keepalive_probes = 10
```

原因是在默认配置下,很多场景的客户端和服务端之间的链路感知时间会很长，导致业务无法在短时间内恢复

如：服务端机器宕机，客户端的 tcp 链路需要经过 net.ipv4.tcp_keepalive_time+net.ipv4.tcp_keepalive_probes*net.ipv4.tcp_keepalive_intvl 时间才会在 TCP 层断链。

7.2.2. 数据库参数配置

1. 数据节点参数配置

表 18 数据节点关键参数

配置项	值	解释	依据和原则
innodb_page_size	默认：16384	Innodb 表空间的页	调整为 64kB，可

	推荐配置：65536	大小。类似于oracle的block。	以兼容更长的联合索引
max_table_record_size	默认：64K 推荐配置：2196608	控制innodb支持的最大行长度开关,支持配置最大行长度为64K、128K、192K	调整为2196608，可以兼容更长的行长度。
innodb_buffer_pool_size	默认：629145600 推荐配置:274877906944	dn个数*此参数=服务器内存*(60~70%)/实例数	结合实际资源情况机型调整
innodb_buffer_pool_instances	默认：8 推荐配置：64	buffer_pool的区域数，是innodb_buffer_pool_size能除尽的数，最大为64。将缓冲池划分为更多的区域,可以提高多个线程并发访问缓冲池的效率,减少争用	调整为64，用于提高业务并行性能。 调整为最大值64即可。
innodb_flush_neighbors	默认：1 推荐配置：0	SSD配置为0，HDD是1。是否把邻近的脏页刷到磁盘,该配置项开启对传统机械硬盘有效果,可以显著减少磁盘寻道	目前服务器硬盘多为SSD，使用0配置即可。

		时间，但对于 SSD 磁盘反而会将写压力集中。	
thread_pool_stall_limit	默认：50 推荐配置：10	将执行的 sql 状态标记为 stall 之前的等待时间,单位为 ms。调小将降低线程池巡检时间影响,可能会创建更多的执行线程。	调整为 10ms，减少线程等待的时间。 10ms 是一个性能较好的实践时间。
temptable_max_ram	默认： 1073741824 推荐配置： 21474836480	执行某些特定查询时,临时表最大可以使用多大字节的内存空间	调整为 20G，可以提高特定查询的执行效率。 一般调整为 20G，对内存无太大影响，且足够使用
temptable_max_mmap	默认： 1073741824 推荐配置： 21474836480	执行某些特定查询时,临时表最大可以使用多大字节的虚拟内存(通过磁盘映射)空间	调整为 20G，可以提高特定查询的执行效率。 一般调整为 20G，对内存无太大影响，且足够使用
thread_pool_sharp_mode	默认： OFF 推荐配置： ON	ON 表示开启线程池 sharp 模式，开启后将会更快地创建线	调整为 ON，提高线程池的响应速度。

		程处理任务 影响:开启后可能会减少线程池等待时间,但是会占用相对多的 CPU 资源。	

2. 计算节点参数配置

表 19 计算节点关键参数

配置项	值	解释	依据和原则
thread_group	默认：2 推荐配置：8	线程组大小，处理数据库返回的响应	调整为 8，可以提高处理性能。
os_message_threshold	默认：50 推荐配置：80	OS 消息队列的限流百分比。大结果集的情况不限速。阈值调大，尽量不限速，保证性能。可能会把网络资源打满	调整为 80%，可以在消息队列中留存更多的消息而不限流,减少消息等待。提高性能。
max_plantreenode_num	默认：300 推荐配置：32	正常调整，一条语句拆分 32 个子语句足够应对非常复杂的多层嵌套子查询的情况	业务语句没有特别复杂的,调整为 32 足以应对所有的语句拆分。
route_instance_number	默认：8 推荐配置：32	路由实例启动个数，处理执行模块发送	调整为 32 个路由实例,并行处理执

		的 sql 请求	行模块发送的 sql 请求，提高性能
exec_thread_num	默认：30 推荐配置：128	执行模块线程数	调整为 128 个执行线程并行处理 sql 请求，提高性能

7.3. 架构调优

1. 修改隔离级别。这种主要针对于一些业务语句对数据一致性不是特别敏感的语句，例如统计类相关的语句。
2. 修改分发规则，让语句下压 DN 执行。这种常见的就是 join 语句，有些数据量特别大的 join 语句，可以考虑修改分片规则，让 SQL 可以整体下推 DN 计算。举例如下：

create table t1(id int primay key, id1 int, id2 int)distributed by hash(id)(g1,g2);

create table t2(id int primay key, id1 int, id2 int)distributed by hash(id1)(g1,g2);

select * from t1 join t2 on t1.id = t2.id;--如上分片规则性能很慢，大量数据传到 CN 层计算

修改 t2 表分片规则为 hash(id)(g1,g2)，这个时候整个 join 语句即可整体下推 DN 进行计算，性能提升会很大

3. 考虑到语句的执行效率，在表设计时建议优先考虑：大表采用多节点

分布存储，小表采用复制表。小表和大表的区分不是根据表的原始数据来判断的，而是根据条件过滤后的数据来判断。

4. 大表和大表的关联，多节点分布存储，采用相同的分发字段和分发策略，关联字段包含分发字段。

7.4. SQL 调优

针对 SQL 层面的调优主要从如下几方面入手：

1. 索引优化，创建合适的索引，如用复合索引代替单列索引。
2. 查询操作尽量带分片键。
3. 语句的等价修改，常见的为 in/not in/exists/not exists 等价改造为 inner join/left join

1) exists/not exists 优化举例

```
select * from t1 where exists (select id from t2 where t1.id = t2.id);
```

如上语句改写为：

```
select t1.* from t1 join (select distinct id from t2)A on t1.id = A.id;
```

(如果 id 值唯一可以去掉 select distinct,直接 t1 与 t2 进行 join;如果 exist 子查询数量较少则无需优化)

注：较少为小于 proxy.ini subwhere_result_line 值

```
select * from t1 where not exists(select id from t2 where t1.id=t2.id);
```

如上语句改写为：

```
select t1.* from t1 left join t2 on t1.id=t2.id where t2.id is null;
```

4. 通过语义进行语句的等价改写。有些语句由于 SQL 写的不好导致性能比较差，这时可以通过调整语句来提升性能，改写的主要目标是减少到 CN 计算的数据，尽量让计算下沉 DN 计算。常见的改写如下，例如：

   Exists 语句改写为 in：

   ```
   select id from t1 where exists(select 1 from t2 where t2.id = t1.id);
   ```

   优化建议语句：

   ```
   select id from t1 where t1.id in(select id from t2);
   ```

   说明：子查询结果集不大于配置项 subwhere_result_line 值，此时可以选择改成 in 优化

5. 查询操作尽量只获取所需的字段，减少因不必要的磁盘读写和网络传输带来性能损耗。

6. 单表查询查询条件尽量控制能使用表的一级索引（即主键索引）。使用索引的性能排序如下：主键索引 >> 唯一索引 >> 普通索引。

7. 分区表涉及的语句尽可能增加分区字段作为 where 条件。

8. 案例简介

截至目前，GoldenDB 已完成 50 余家金融机构重要业务系统分布式架构改造，是业界唯一一家实现全面覆盖国有大行、股份制行和头部券商核心业务交易系统数据库替代的信创数据库。6 家国有大行核心业务已进入

3 家，12 家股份制银行核心业务已进入 8 家，农信联社核心业务已进入 5 家，城商农商行核心业务超过 10 家,其他关键业务突破超过 20 家。前 10 大证券公司已进入 4 家。

GoldenDB 金融行业整体案例已覆盖国有大行、政策性行、股份制行、农信联社、城商行和农商行全系列银行、证券交易所、券商及保险机构。

8.1. 建设银行对私核心业务系统项目

建设银行对私核心系统先后于 2021 年 12 月和 2022 年 12 月完成两省及八省投产，并于 2023 年 11 月 18 日完成整体投产，打造业界首个国有大行主机核心业务分布式改造的标杆。系统采用 4 套 GoldenDB 集群，承载 8 亿用户，21 亿账户，采用 2 地 4AZ 的多活部署，可靠性能力进一步提升，设计性能百万 TPS，端到端的平均交易时延为 53ms。在本项目中，集群内部的分布式事务由 GoldenDB 解决，应用只需要解决跨集群的分布式事务，极大缓解了应用侧的压力。

8.2. 中信银行核心业务系统项目

自 2014 年起，GoldenDB 先后在中信银行冠字号系统、金融同业平台、零售客户统一积分等多个业务投产。通过全面验证后，GoldenDB 于 2019 年 10 月在信用卡核心投产，承载 1.1 亿账户，平均交易时延 36ms。2020 年 5 月在账务核心投产，至今已稳定运行超 3 年，可靠性达 6 个 9,，系

统承载 3 亿用户、15 亿账户，日均交易 3 亿笔，数据分片采用 HASH 算法，事务一致性由数据库承担，平均交易时延 50ms，完全满足人民银行对交易时延的要求。打造大型商业银行首个核心业务改造标杆。

8.3. 赣州银行核心业务系统项目

2020 年 10 月，赣州银行信贷及核算核心基于 GoldenDB 投产，打造首个完成核心业务分布式改造的试点城商行。项目采用多应用共用数据库集群，实现快速部署、提升资源利用率。充分发挥分布式架构的优势，跑批性能大幅提升，远超原 DB2 的性能，达到预期设计标准。

8.4. 贵州银行核心业务系统项目

2020 年 11 月，贵州银行核心业务系统基于 GoldenDB 投产，同步投产的还有 ECIF、全渠道、全媒体、信贷、非零内评等核心及关键业务平台，是首个基于分布式架构完成核心在内的多业务同时投产的城商行。项目通过 GoldenDB 分布式事务一致性降低业务改造量、通过两地三中心架构保障系统高可用。

8.5. 南京银行理财子业务业务系统项目

2021 年 11 月，南京银行理财子系统基于 GoldenDB 投产，凭借对 Oracle 的高兼容性，GoldenDB 协助业务实现平滑迁移，满足批处理和联

机交易不同的使用需求，跑批性能提升 30%。此外，GoldenDB 也已在中间件平台、对公 ECIF 等一类系统和其它十余套二类、三类系统中实施，陆续将有 50 套系统完成投产。

8.6. 徽商银行互联网核心业务系统项目

2023 年 3 月，徽商银行互金核心基于 GoldenDB 投产，采用同城双中心架构，支撑业务双活部署。承载 5000 万账户、性能达 3000TPS。具备线性扩展能力，满足未来业务发展需要。

8.7. 国泰君安新一代核心交易系统项目

2022 年 8 月，国泰君安新一代低延时核心交易系统基于 GoldenDB 投产，成为业界首个证券行业核心业务分布式改造案例。项目采用 HTAP 架构并行计算能力，解决券商场景复杂 SQL 查询；通过批量插入协议：满足数据上场性能要求。方案可支持账户数量 5000w，盘中交易写入 60w 行/s，数据上场写入性能达 792w 行/s。